UNIVERSITI TEKNOLOGI MARA

A LIGHTWEIGHT CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE FOR IMAGE CLASSIFICATION ON EDGE DEVICES

MUHAMMAD ABBAS
ABU TALIB

MSc

September 2026

UNIVERSITI TEKNOLOGI MARA

A LIGHTWEIGHT CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE FOR IMAGE CLASSIFICATION ON EDGE DEVICES

MUHAMMAD ABBAS ABU TALIB

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science
(Electrical Engineering)

Faculty of Electrical Engineering

September 2026

CONFIRMATION BY PANEL OF EXAMINERS

I certify that a Panel of Examiners has met on 7 July 2026 to conduct the final examination of Muhammad Abbas Abu Talib on his Masters of Science thesis entitled "A Lightweight Convolutional Neural Network Architecture for Image Classification on Edge Devices" in accordance with Universiti Teknologi MARA Act 1976 (Akta 173). The Panel of Examiners recommends that the student be awarded the relevant degree. The Panel of Examiners was as follows:

Muhammad Nabil Hidayat, PhD
Associate Professor
Faculty of Applied Sciences
Universiti Teknologi MARA
(Chairman)

Muhammad Khusairi Osman, PhD
Associate Professor
Faculty of Electrical Engineering
Universiti Teknologi MARA
(Internal Examiner)

Fakroul Ridzuan Hashim, PhD
Associate Professor
Faculty of Engineering
University Pertahanan Nasional
Malaysia
(External Examiner)

PROF DR HJH ZURAEDA BINTI IBRAHIM
Dean
Institute of Postgraduates Studies
Universiti Teknologi MARA

Date: 15 September 2026

AUTHOR'S DECLARATION

I declare that the work in this thesis was carried out in accordance with the regulations of Universiti Teknologi MARA. It is original and is the results of my own work, unless otherwise indicated or acknowledged as referenced work. This thesis has not been submitted to any other academic institution or non-academic institution for any degree or qualification.

I, hereby, acknowledge that I have been supplied with the Academic Rules and Regulations for Postgraduate, Universiti Teknologi MARA, regulating the conduct of my study and research.

Name of Student	:	Muhammad Abbas Abu Talib
Student ID. No.	:	2024740247
Programme	:	Master of Science (Electrical Engineering) – CEEE750
Faculty	:	Electrical Engineering
Thesis Title	:	A Lightweight Convolutional Neural Network Architecture for Image Classification on Edge Devices
Signature of Student	:	……..
Date	:	September 2026

ABSTRACT

A lightweight Convolutional Neural Network (CNN) has become one of the major studies in the machine learning field to optimise its potential for employing it on resource-constrained devices. However, achieving stable, robust, and practical usage of edge computing still poses a critical challenge, particularly because the benchmark for fair comparison in terms of accuracy-efficiency trade-off and performance evaluation of lightweight CNN models across different hardware platforms is still lacking in generalisation. For that, this research aims to optimise a lightweight convolutional neural network model tailored specifically for deployment on edge devices for image classification tasks. To that end, several models were extensively investigated from the perspective of model training, validation and inference. Specifically, MobileNetV2, MobileNetV3, ShuffleNetV2, and VGG-16 were evaluated under both transfer learning and without transfer learning conditions. Among these, ShuffleNetV2 was observed to have the most consistent and stable performance as a recent lightweight CNN model among the investigated architectures. Based on the identified model, post-training channel pruning and network simplification techniques were applied to reduce the number of filters and overall computational complexity. As a result, the experiment shows that the optimised ShuffleNetV2 maintained comparable classification accuracy based on the different use cases of the image classification tasks deployed in different edge devices. Specifically, the model sustained comparable accuracy at approximately 20% sparsity level with 83.86% accuracy and 33.49 ms inference time on the small-scale dataset (i.e., the TrashNet dataset), 30% sparsity level with 91.64% accuracy and 16.57 ms inference time on the medium-scale dataset (i.e., the ball screw drive dataset), and up to 80% sparsity level with 91.24% accuracy and 25.01 ms inference time on the large-scale dataset (i.e., the concrete surface crack dataset) for the optimised model deployed in the Raspberry Pi 5. Additionally, the edge deployment evaluations indicate that the inference time performance on the Raspberry Pi 5 achieved a lower inference time compared to the NVIDIA Jetson Nano, which suggests that the proposed optimisation strategies are more compatible with CPU-based hardware. These findings highlight the importance of hardware-aware optimisation and motivate further research to adapt these strategies for other AI accelerators such as GPUs, TPUs, and NPUs.

ACKNOWLEDGEMENT

Firstly, I wish to express my deepest gratitude to The Almighty God, Allah S.W.T. because of His Gracious and Blessings, for giving me the opportunity to embark on my Master and for completing this long and challenging journey successfully.

My gratitude and thanks go to my supervisor Prof. Madya Ir. Ts. Dr. Samsul Setumin for his great understanding and continuous support with knowledge, guidelines, and utilities. My appreciation also goes to the co-supervisors Dr. Adi Izhar Che Ani and Dr. Siti Juliana Abu Bakar for additional advice and extra support.

Special thanks to my family, friends, and everyone who was directly or indirectly involved in supporting me throughout this project. Alhamdulillah.

TABLE OF CONTENTS

		Page
CONFIRMATION BY PANEL OF EXAMINERS		**ii**
AUTHOR'S DECLARATION		**iii**
ABSTRACT		**iv**
ACKNOWLEDGEMENT		**v**
TABLE OF CONTENTS		**vi**
LIST OF TABLES		**ix**
LIST OF FIGURES		**xi**
LIST OF ABBREVIATIONS		**xiv**
CHAPTER 1 INTRODUCTION		**1**
1.1	Research Background	1
1.2	Problem Statement	4
1.3	Research Objectives	5
1.4	Significance of the Study	6
1.5	Thesis Scope	8
1.6	Thesis Outline	8
CHAPTER 2 LITERATURE REVIEW		**10**
2.1	Introduction	10
2.2	Lightweight CNN Architectures	10
2.3	Types of CNN	14
2.4	Models Application	15
2.5	Type of Edge Devices	15
2.6	Evaluation Types and Metrices	16
2.7	Performance Comparison	17
2.8	Summary	19

CHAPTER 3 RESEARCH METHODOLOGY **24**
3.1 Introduction 24
3.2 Research Methodology Overview 24
3.3 Stage 1: Investigate the Current Lightweight CNN Models that Best for Image Classification Task on Edge Device 28
3.3.1 Lightweight CNN Models Selection 28
3.3.2 Performance Evaluation for Selected Models 29
3.3.3 The Best Performed Model Identification 35
3.4 Stage 2: Optimise the Performance of the Identified Lightweight CNN Model using Appropriate Optimisation Strategies 36
3.4.1 Lightweight CNN Architecture Optimisation Strategies 38
3.4.2 Evaluate the Testing Performance of the Optimised Model on Desktop Computer Platform 40
3.5 Stage 3: Evaluate the Performance of the Optimised Lightweight CNN Model in terms of Classification Accuracy and Inference Time on Edge Devices 42
3.5.1 Deploy the Optimised Model trained with several Image Classification Datasets on Edge Devices 43
3.5.2 Assess Comparative Analysis for The Optimised Model Performance Evaluation 47
3.6 Summary 47

CHAPTER 4 RESULTS AND DISCUSSION **49**
4.1 Introduction 49
4.2 Identification of the Best Performed Current Lightweight CNN Models for Image Classification Task on Edge Devices 49
4.2.1 Training and Validation Result using Transfer Learning for the Selected Models 50
4.2.2 Training and Validation Result without using Transfer Learning for the Selected Models 55
4.2.3 Inference Result of the Transfer Learning Models 60
4.2.4 Inference Result of the Models without Transfer Learning 62
4.2.5 Performance comparison for the Current Lightweight CNN Models 64

4.3 Optimised the Performance of the Identified Lightweight CNN Models using Appropriate Optimisation Strategies 65
4.3.1 Inference Performance Evaluation of the Optimised Models on a Desktop Computer 66
4.3.2 Comparative Analysis between the Original Model and the Optimised Models 73
4.4 Performance Evaluation of the Optimised Lightweight CNN Model on Edge Devices using Appropriate Evaluation Metrics 75
4.4.1 Inference Performance Evaluation on Jetson Nano and Raspberry Pi 5 76
4.4.2 Comparative Analysis in terms of Classification Accuracy and Inference Time 88
4.5 Conclusion 90

CHAPTER 5 CONCLUSION **92**
5.1 Conclusion 92
5.2 Research Contribution 93
5.3 Future Work 94
REFERENCES **95**
APPENDICES **101**
AUTHOR'S PROFILE **112**

LIST OF TABLES

Tables	Title	Page
Table 2.1	Research Database Literature Classifications	19
Table 2.2	Research Database Edge Devices' Performance Classifications	22
Table 3.1	Groundnut Plant Leaf Disease Dataset Specifications	31
Table 3.2	Desktop Computer Platform Configurations	32
Table 3.3	Training Hyperparameter for the Selected Model	32
Table 3.4	NVIDIA Jetson Nano Platform Configurations	35
Table 3.5	TrashNet Dataset Specifications	44
Table 3.6	Ball Screw Drive Dataset Specifications	45
Table 3.7	Concrete Surface Dataset Specifications	45
Table 3.8	Raspberry Pi 5 Platform Configurations	47
Table 4.1	Jetson Nano Classification Accuracy and Inference Time using Transfer Learning	60
Table 4.2	Jetson Nano Classification Accuracy and Inference Time without using Transfer Learning	64
Table 4.3	Inference Results of ShuffleNetV2x0.5 Characteristics and Performances on Desktop Computer	67
Table 4.4	Inference Results of ShuffleNetV2x1.0 Characteristics and Performances on Desktop Computer	68
Table 4.5	Inference Results of ShuffleNetV2x1.5 Characteristics and Performances on Desktop Computer	70
Table 4.6	Inference Results of ShuffleNetV2x2.0 Characteristics and Performances on Desktop Computer	71

Table 4.7	Performance Comparison of ShuffleNetV2 Variants: Original vs. Optimised Model	73
Table 4.8	Performance Comparison of ShuffleNetV2 Variants based on Image Classification Task of Groundnut Plant Leaf Diseases Dataset	84
Table 4.9	Performance Comparison of ShuffleNetV2 Variants based on Image Classification Task of Ball Screw Drive Dataset	85
Table 4.10	Performance Comparison of ShuffleNetV2 Variants based on Image Classification Task of TrashNet Dataset	86
Table 4.11	Performance Comparison of ShuffleNetV2 Variants based on Image Classification Task of Concrete Surface Crack Dataset	87
Table 4.12	Performance Comparison of ShuffleNetV2 Variants: Original vs. Optimised Model	88

LIST OF FIGURES

Figures	Title	Page
Figure 1.1	The Interrelation of Data Science to Artificial Neural Networks through AI, Machine Learning, and Deep Learning	1
Figure 1.2	The Basic CNN Architecture and its Training Process	4
Figure 2.1	Several Different Common Lightweight CNN Architectures	11
Figure 2.2	MobileNetV1 (i.e., MobileNet) and MobileNetV2 Comparisons	11
Figure 2.3	MobileNetV3 Basic Convolution Operation	12
Figure 2.4	ShuffleNetV1 (i.e., (a) and (b)) and ShuffleNetV2 (i.e., (c) and (d)) Comparisons	13
Figure 2.5	Basic VGG-16 Architecture	14
Figure 2.6	The Common Types of CNNs	15
Figure 2.7	Various Applications of Lightweight CNN Use for Different Sectors of Industry	16
Figure 2.8	Some of the General Edge Devices Used to Embed a Lightweight CNN Architecture	17
Figure 2.9	(a) The Evaluation Types and (b) the Evaluation Metrices Used Frequently in the Recent Studies.	18
Figure 2.10	Performance Comparison in terms of Average Accuracy for Each Edge Device	18
Figure 3.1	(a) Flowchart of this Research Methodology Correspond to Research Objective 1	25
Figure 3.1	(b) Flowchart of this Research Methodology Correspond to Research Objective 2	26

Figure 3.1	(c) Flowchart of this Research Methodology Correspond to Research Objective 3	27
Figure 3.2	Subprocess of Lightweight CNN Models Selection	30
Figure 3.3	(a) Subprocess for Selected Models' Performance Evaluation with Transfer Learning	33
Figure 3.3	(b) Subprocess for Selected Models' Performance Evaluation without Transfer Learning	34
Figure 3.4	Subprocess for the Best Performed Model Identification	37
Figure 3.5	Subprocess for Optimising the Identified Lightweight CNN Architecture	39
Figure 3.6	Subprocess to Train, Optimise, and Deploy the Additional Image Classification Tasks	44
Figure 3.7	Subprocess for Evaluating All the Findings	48
Figure 4.1	Training Loss vs. Epochs using Transfer Learning	51
Figure 4.2	Validation Loss vs. Epochs using Transfer Learning	52
Figure 4.3	Training Accuracy (%) vs. Epochs using Transfer Learning	53
Figure 4.4	Validation Accuracy (%) vs. Epochs using Transfer Learning	54
Figure 4.5	Training Loss vs. Epochs without using Transfer Learning	56
Figure 4.6	Validation Loss vs. Epochs without using Transfer Learning	57
Figure 4.7	Training Accuracy (%) vs. Epochs without using Transfer Learning	58
Figure 4.8	Validation Accuracy (%) vs. Epochs without using Transfer Learning	59
Figure 4.9	Sparsity Level (%) vs. Inference Time on CPU and GPU of ShuffleNetV2x0.5	68

Figure 4.10 Sparsity Level (%) vs. Inference Time on CPU and GPU of ShuffleNetV2x1.0 69

Figure 4.11 Sparsity Level (%) vs. Inference Time on CPU and GPU of ShuffleNetV2x1.5 71

Figure 4.12 Sparsity Level (%) vs. Inference Time on CPU and GPU of ShuffleNetV2x2.0 72

Figure 4.13 Inference Time on CPU and GPU vs. Sparsity Level (%) of the Optimisation on Groundnut Plant Leaf Diseases Classification Task 77

Figure 4.14 Inference Time on CPU and GPU vs. Sparsity Level (%) of the Optimisation on Ball Screw Drive Defect Classification Task 79

Figure 4.15 Inference Time on CPU and GPU vs. Sparsity Level (%) of the Optimisation on TrashNet Classification Task 80

Figure 4.16 Inference Time on CPU and GPU vs. Sparsity Level (%) of the Optimisation on Concrete Surface Crack Classification Task 82

LIST OF ABBREVIATIONS

Abbreviations

Adagrad	Adaptive Gradient
AdamW	Adaptive Moment Estimation with Weight Decay
AI	Artificial Intelligence
API	Application Programming Interface
CL	Confidence Level
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FLOPs	Floating Point Operations
FN	False Negatives
FP	False Positives
FPGA	Field-Programmable Gate Array
GB	Gigabytes
GP	General Purpose
GPU	Graphics Processing Unit
ID	Identification Number
IoT	Internet of Things
LBL	Label
MACs	Multiply-Accumulate Operations

MB	Megabytes
MCU	Microcontroller
MFG	Manufacturing
ms	Milliseconds
NAS	Neural Architecture Search
NetAdapt	Network Adaptation
NPU	Neural Processing Units
ONNX	Open Neural Network Exchange
OS	Operating System
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue
RMSprop	Root Mean Square Propagation
SDK	Software Development Kit
SE	Squeeze-and-Excitation
SGD	Stochastic Gradient Descent
SoC	System-on-a-Chip
TN	True Negatives
TP	True Positives
TPU	Tensor Processing Units

CHAPTER 1
INTRODUCTION

1.1 Research Background

Researchers these days are primarily focused on the advancement of Artificial Intelligence (AI) technology in order to enhance society's quality of life and facilitate the industrial revolution. Although this discipline was introduced back in the 1950s, it has gone through rapid development in the past decades, which has covered both inside and outside of the computer science field (Choi et al., 2020; N. Sharma et al., 2021). It can be seen that many technologies and non-technology-based journals have published articles related to AI (Choi et al., 2020; N. Sharma et al., 2021). AI has progressed from simple rule-based systems to more complicated algorithms that can make autonomous decisions and solve problems. The primary idea underlying AI is to develop systems capable of doing activities that would normally need human intellect, such as visual perception, speech recognition, decision-making, and language translation (Choi et al., 2020; N. Sharma et al., 2021). Figure 1.1 shows the interrelation of data science to artificial neural networks through AI, machine learning, and deep learning (Choi et al., 2020).

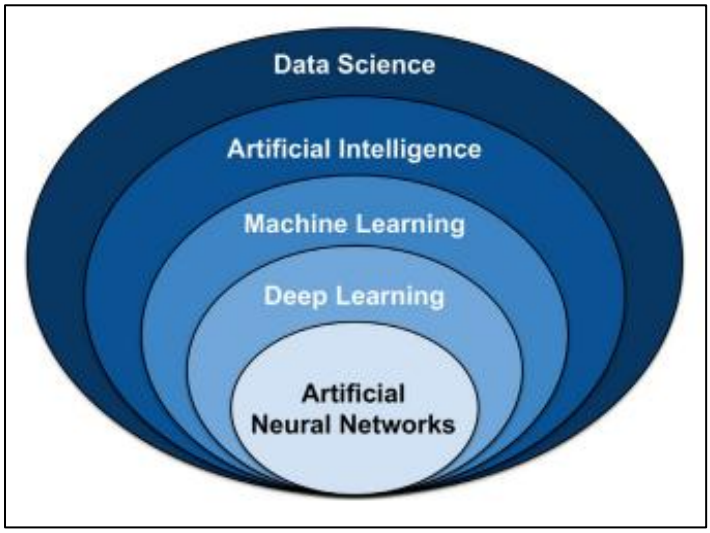

Figure 1.1 The Interrelation of Data Science to Artificial Neural Networks through AI, Machine Learning, and Deep Learning

Sources: (Choi et al., 2020; N. Sharma et al., 2021)

Furthermore, machine learning is one of the most common subfields in AI, where it takes a different approach from a classical programming method. So, instead of using an algorithm for a specific problem or function, machine learning uses a certain dataset for its algorithm to learn, predict, and decide the outcome (Choi et al., 2020; N. Sharma et al., 2021). Primarily, machine learning is typically categorized into four major types such as supervised learning, which involves training models on labelled data, unsupervised learning, which involves searching for patterns in unlabelled data, semi supervised learning, which includes both supervised and unsupervised learning, and reinforcement learning, which teaches models to make decisions based on trial and error (Choi et al., 2020; N. Sharma et al., 2021). In addition, the application of machine learning is commonly divided into object classification or regression (i.e., prediction). Some typical examples of algorithms in machine learning include artificial neural networks, decision trees, linear regression, and support vector machines (Choi et al., 2020; N. Sharma et al., 2021).

Moreover, Convolutional Neural Network (CNN) is one of machine learning's artificial neural network algorithms that is specialised for image-based tasks (Choi et al., 2020; Hou et al., 2022; Naveen et al., 2021; Stahl et al., 2021; Yamashita et al., 2018). In other words, CNN is fundamental in many computer vision tasks such as image detection, recognition, classification, regression, and segmentation. CNN is made up of three main layers, including convolutional, pooling, and fully connected layers (Choi et al., 2020; Hou et al., 2022; Naveen et al., 2021; Stahl et al., 2021; Yamashita et al., 2018). Figure 1.2 depicts the basic CNN architecture and its training process (Choi et al., 2020; Hou et al., 2022; Naveen et al., 2021; Stahl et al., 2021; Yamashita et al., 2018). First, convolutional layers apply filters to incoming data, capturing spatial hierarchies and local patterns that are necessary for applications such as image identification (Choi et al., 2020; Hou et al., 2022; Naveen et al., 2021; Stahl et al., 2021; Yamashita et al., 2018). Second, pooling layers lower the dimensionality of the data, increasing computing efficiency and resilience (Choi et al., 2020; Hou et al., 2022; Naveen et al., 2021; Stahl et al., 2021; Yamashita et al., 2018). Third, fully connected layers make high-level decisions based on the extracted characteristics (Choi et al., 2020; Hou et al., 2022; Naveen et al., 2021; Stahl et al., 2021; Yamashita et al., 2018). Simply put, the first two main layers perform feature extraction from the input data and the third main layer maps the extracted features to decide or predict the output

data (Choi et al., 2020; Hou et al., 2022; Naveen et al., 2021; Stahl et al., 2021; Yamashita et al., 2018).

Nevertheless, the development of CNN's applications usually involves big data, which relies heavily on cloud infrastructure and resources for high computation complexity, memory and power consumption (Baller et al., 2021; Hou et al., 2022; J. Liu et al., 2020; Naveen et al., 2021; Stahl et al., 2021). In recent years, the widespread use of cloud computing in many fields of CNN applications has raised some concern regarding strict latency requirements, strained network capacity, as well as privacy and security issues (Baller et al., 2021; Hou et al., 2022; J. Liu et al., 2020; Naveen et al., 2021; Stahl et al., 2021). Ultimately, in order to overcome these problems and optimise CNN's applications, an increasing demand for deploying deep learning models directly onto edge devices to enable real-time inference and decision-making has been introduced. Edge computing includes processing data at or near the source of data creation, which is called edge devices, such as Internet of Things (IoT) devices, smartphones, or sensors, rather than using a centralised cloud infrastructure.

In addition, conventional CNN architectures are typically designed to maximize accuracy without taking into account the computational limitations of edge devices. These models generally have millions of parameters, require large amounts of memory and have long inference times that are beyond the capabilities of resource-constrained hardware. Therefore, to solve these issues, lightweight CNNs have been introduced to make a trade-off between the classification accuracy and the hardware capabilities. A lightweight CNN usually refers to the reduced number of parameters and computational complexity compared to a conventional CNN. A lightweight CNNs enable faster inference, lower memory usage, and reduced power consumption. Consequently, they are well suited for deployment on edge devices in order to allow for real-time applications without relying on high-performance GPUs or cloud-based computing resources.

Following this, optimization techniques are applied to further improve the efficiency of lightweight CNN models by reducing unnecessary computations and model complexity. Methods such as pruning and network simplification can help reduce model size while maintaining acceptable accuracy. However, the effectiveness of these optimization methods depends on the target hardware, as different devices have different computational capabilities and limitations. For example, a CPU-based hardware typically employs a sequential computing, while a GPU-based hardware

features a parallel computing methods for which highly likely affecting optimization technique for a lightweight model. Therefore, the design of lightweight CNN models requires consideration of both model performance and hardware constraints to achieve efficient deployment on edge devices.

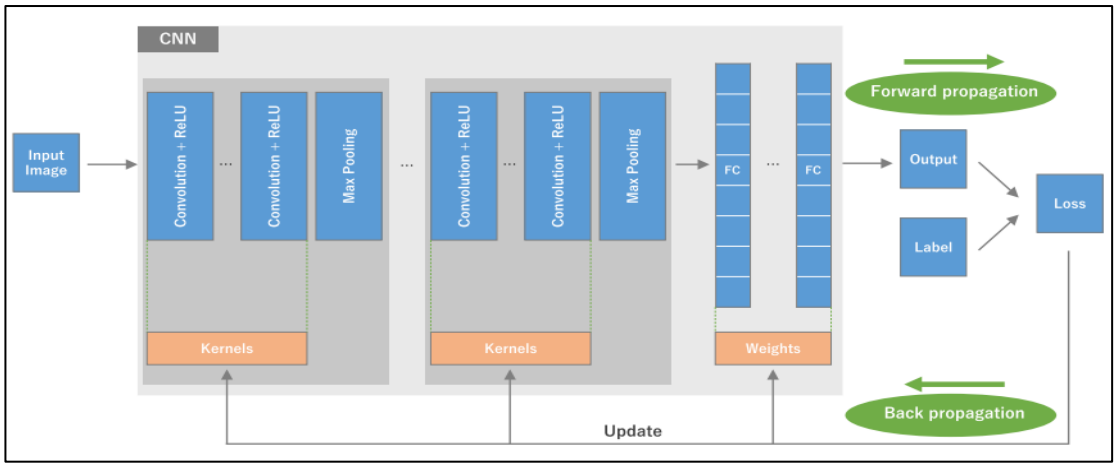

Figure 1.2 The Basic CNN Architecture and its Training Process

Source: (Yamashita et al., 2018)

1.2 Problem Statement

Edge computing can become a breakthrough solution in replacing certain applications of AI, where strict latency requirements while maintaining and sustaining high accuracy, are needed. However, optimising efficiency for edge deployment poses a critical challenge due to limited computational resources and power constraints. According to the research in (F. Wang et al., 2020), several research challenges, such as achieving stable, robust, and practical usage of edge computing, require non-stop effort and continuous study from the perspectives of model training, model inference, and application enhancement, as well as hardware and software optimisation. Specifically, thermal constraints (e.g., slow heat transfer) and processing constraints (e.g., sequential and parallel computing) can increase latency and thus affecting the reliability of edge devices. Considering that the subject is still in the early stage of development, this research project aims to optimise a CNN model tailored specifically for deployment on edge devices for image classification tasks.

Furthermore, as many researchers have employed edge computing to gain various benefits, most studies in this domain have used different methods of design and

modifications to produce a lightweight CNN for edge device deployment. However, the benchmark for a fair comparison and generalisation in order to find the best lightweight CNN model is still missing since the applications vary from each other (Hussain et al., 2022; Kamath & Renuka, 2023; J. Liu et al., 2020; F. Wang et al., 2020; Zaidi et al., 2022). To address this issue, the observation from several aspects, such as model training and model inference, as mentioned in (F. Wang et al., 2020), can provide more insight and understanding in order to compare the performance between different lightweight CNN models. Thus, a thorough investigation needs to be conducted to find out the current best-performing lightweight CNN model among a few recently and majorly utilised for edge applications.

Besides, although there are many lightweight CNN models that have been proposed by many researchers, most of their accuracy varies across classification models and datasets used. Not only that, there are several other accuracy-efficiency trade-offs, such as the number of parameters, floating-point operations (FLOPs), model size, and hardware accelerators (i.e., CPU and GPU) that also affect the performance of lightweight CNN models. For example, the classification task using MobileNet in (B. Wang et al., 2021) shows the accuracy of 74.5%, while in (Y. Liu et al., 2023), this particular model produced an accuracy of 97.13%. Based on this example, it clearly shows a huge discrepancy in accuracy, even though the same model (i.e., MobileNet) is used for the classification task. With that, from the point of hardware and software optimisation for lightweight CNN architectures (F. Wang et al., 2020), architecture optimisation techniques (i.e., channel pruning, network simplification, and sparsity level optimisation) can be applied to acquire high and consistent performance across various image classification datasets. Hence, several optimisation strategies will be explored and used on the current best-performing lightweight CNN model to further improve its performance as part of application enhancement, as stated in (F. Wang et al., 2020).

1.3 Research Objectives

This research aims to optimise a lightweight convolutional neural network model tailored specifically for deployment on edge devices for image classification tasks. In order to achieve this aim, several objectives are considered, such as:

a) To investigate the current lightweight CNN models that best for image classification task on edge device.
b) To optimise the performance of the identified lightweight CNN model using appropriate optimisation strategies.
c) To evaluate the performance of the optimised lightweight CNN model in terms of classification accuracy and inference times on edge devices.

1.4 Significance of the Study

This research generally focuses on the role of lightweight CNN architectures in providing efficient image classification inference models that can be directly deployed on edge devices, without the need for cloud infrastructure implementations. With that, the expansion of edge devices, such as smartphones, drones, IoT sensors, and embedded systems, has empowered edge computing with the development of lightweight CNN architectures that are capable of operating efficiently on devices with limited compute power, memory, and energy. While one of the reasons for this research being conducted is to gain insight into the feasibility and capability of lightweight CNN on edge devices, some of the key significances of this study are highlighted as follows:

i) Reliability of Edge Devices: Edge devices often have minimal computational power, memory, and storage. Unlike high-end cloud infrastructure, these devices may simply have a few Gigabytes (GB) of Random Access Memory (RAM) and a basic Central Processing Unit (CPU) or Graphics Processing Unit (GPU). Lightweight CNNs, which have fewer parameters and operations, can be deployed on these devices to save memory during model loading and inference. As these models mostly have smaller file sizes, it makes them easier to store and share on devices with limited flash or disk storage capacity.
ii) Secure and Local Data Usage: Edge devices can ensure that data is processed locally, which improves privacy and security by storing sensitive data (e.g., face recognition or medical imaging) on the device. Also, it can be utilised in locations with limited or no internet access, such as rural farms, disaster zones, or offshore areas. By that, this application can guarantee continuous operation even in faraway areas or in crucial situations by eliminating the need to send raw data to the cloud for processing. Hence, edge devices can secure data

privacy, save significant network bandwidth, and achieve real-time inference with low latency.

iii) Cost-Effective in Advancing AI Accessibility: Lightweight CNNs can run on low-cost devices, making AI solutions affordable for startups, educational institutions, and small-scale deployments. This notion enables the integration of powerful AI into everyday devices such as smartphones, wearable devices, and home appliances. By performing inference locally on edge devices, it minimises the need to use cloud infrastructure and thus lowers the operational expenses. Therefore, the benefits of AI can be universalised and more available to end-consumers worldwide, regardless of access to high-end technology.

iv) Environmental Impact: While lightweight CNNs use significantly less power to operate due to edge devices' constraints, it makes them perfect for low-powered gadgets and contribute to energy sustainability. By decreasing processing load, lightweight CNNs contribute to decreasing the carbon footprint of AI applications, making them more advantageous for the worldwide applications of energy-efficient initiatives. Also, the ability to run inference models on low-end equipment increases their usefulness in minimising electronic waste and the environmental impact of regular hardware updates.

In summary, edge computing is important as one of the AI technology breakthroughs that could replace cloud computing for certain applications where inference time, data privacy, or cost-effective solutions are necessary. By that, Lightweight CNN architectures are a cornerstone for edge computing because they need to handle edge devices' limits while also unleashing their potential for AI applications. Moreover, the outcome of this research can be published by delivering the optimised and improved lightweight CNN architecture through open-source platforms (e.g., GitHub, Hugging Face, and others). Thus, developers, researchers, or any kind of users can develop their own image classification model, particularly for their own applications, by using edge devices. All in all, edge computing is and will be useful in inspiring more innovation in technology, ranging from healthcare and agriculture to smart cities and industrial automation.

1.5 Thesis Scope

This research proposes to study lightweight CNN architectures for the image classification task on resource-constrained devices. For that, the research will be conducted that revolves around two major aspects, which are software and hardware, in order to realise the research objectives. As such, the software part will be mostly included in developing image classification inference models, while the hardware part will involve configuring the edge device features and limitations. With the expected outcome to close the gap for edge computing practical applications, this study will thoroughly explore, analyse, develop, and validate several factors that are related to the scope of this study, such as:

i) Architectures/Models: Lightweight CNN architectures available in PyTorch, specifically MobileNetV2, MobileNetV3 (i.e., Small and Large), ShuffleNetV2 (i.e., x0.5, x1.0, x1.5, x2.0), and VGG-16.
ii) Datasets: Image classification datasets of varying sizes - Groundnut plant leaf disease (i.e., 6 classes, 7910 images), TrashNet (i.e., 6 classes, 2527 images), Ball screw drive (i.e., 2 classes, 21835 images), and Concrete surface crack (i.e., 2 classes, 96092 images).
iii) Edge Devices: NVIDIA Jetson Nano (i.e., GPU-based) and Raspberry Pi 5 (i.e., CPU-based).
iv) Programming / Scripting: Python with PyTorch framework.

1.6 Thesis Outline

As a summary of this chapter, firstly, some knowledge background was overviewed in the first subchapter. After that, the next subchapter highlighted the problem statement of this research based on several recent studies. Following that, the aim and three primary objectives were defined to provide a solution to the problem statement. Next, the scope of this study implied the covered area and the related field of this research. Finally, the significance of this study was declared to reflect the importance of this research and its benefits from different perspectives.

Additionally, in order to acquire a better understanding of this research, the second chapter will explain the recent literature and studies that are related to this

research. It follows several subchapters which described some of the major lightweight CNN architectures, the types of CNN, the applications of CNN models, the types of edge devices, and the evaluation types and metrics. Moreover, a brief performance comparison will be shown and analysed in order to get a better idea of the practical side of the research. Last but not least, the second chapter will be concluded with a summary, which includes the research gaps.

After that, the third chapter will discuss the methodology of this research. It will depict the flows of this research that was based on the research objectives. In the first subchapter, it will focus more on the ways to investigate and identify the best current lightweight CNN for the image classification task on an edge device. Then, the constructed optimisation strategies and several evaluation metrics to measure the feasibility, robustness, and effectiveness of the optimised model will also be discussed. For the third subchapter, it will focus more on describing the experimental setup of the optimised model on real, limited-resource hardware.

Next, chapter four will observe, analyse, and discuss all the results collected based on the methodology to achieve all the respective research objectives. In detail, the first subchapter will describe the intensive results that were collected to investigate and identify the current best-performing lightweight CNN model. The second subchapter will follow to acquire several results of the optimisation strategies applied to the identified model. Then, the last subchapter will complete the results and justification with the data of the optimised model from various image classification tasks deployed in different resource-constrained devices.

In the end, the research will be concluded in chapter five, which will summarise all the findings for the research objectives with the respective methodology. Finally, several insights and recommendations for a continuous research scope for future study are also included.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

In this literature section, the reviewed articles were classified according to the lightweight CNN architectures for the purpose of edge device implementation. It comprises all the lightweight CNN architectures or based models, the types of CNN, and its applications that have been researched in recent years. It also focuses on the overview of lightweight CNN architectures that have been experimented on edge devices, which includes its performance evaluation in terms of their evaluation types, metrices, and performance comparison in terms of average accuracy.

2.2 Lightweight CNN Architectures

Firstly, Figure 2.1 shows the number of architectures used from all the research in Table 2.1. Based on the architectures or based model, it can be seen that several similar networks were being used as their approaches, such as MobileNet with 9 (19.1%), ShuffleNet with 4 (8.5%), VGG16 with 4 (8.5%), CondenseNet with 3 (6.4%), etc., respectively, in terms of the number of their usage. Meanwhile, the Others with 21 (44.7%) represent the number of different models with only one usage. All these are some of the state-of-the-art approaches that are currently being used by researchers in this field. While some of the researchers use them as benchmarks, there are also several others that modify these original networks with various versions to improve their performance. Aside from that, there were a few with hybrid models or multimodal which combined two or more networks together by using novel approaches. For example, research in (X. Li et al., 2023) combined a lightweight CNN architecture's MobileNet model with a Vision Transformer architecture. All in all, MobileNet, including its various version is the most used lightweight CNN architecture for edge devices implementation.

In detail, the MobileNet architecture was designed to enhance mobile and embedded computer vision models by introducing depth-wise separable convolutions (i.e., a depthwise convolution followed by a pointwise convolution). This design emphasises that the computational cost can be reduced by splitting the conventional

convolutional method into two different operations, such as with depthwise convolution focus on spatial dimensions (i.e., height and width), while pointwise convolution focuses on channel dimensions. Following that, several architectural improvements were introduced in MobileNetV2, as shown in Figure 2.2.

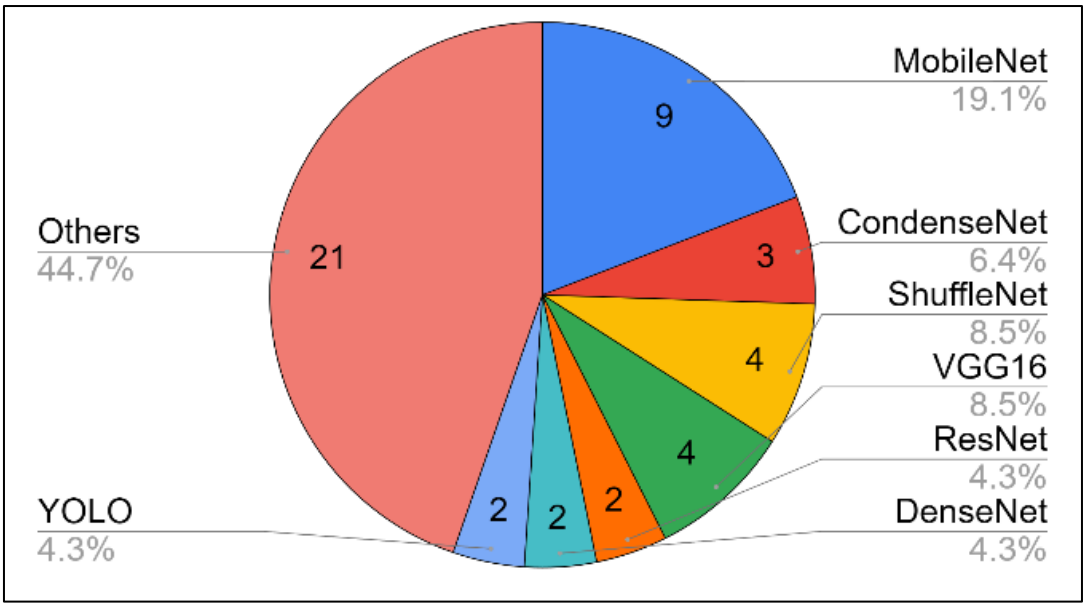

Figure 2.1 Several Different Common Lightweight CNN Architectures

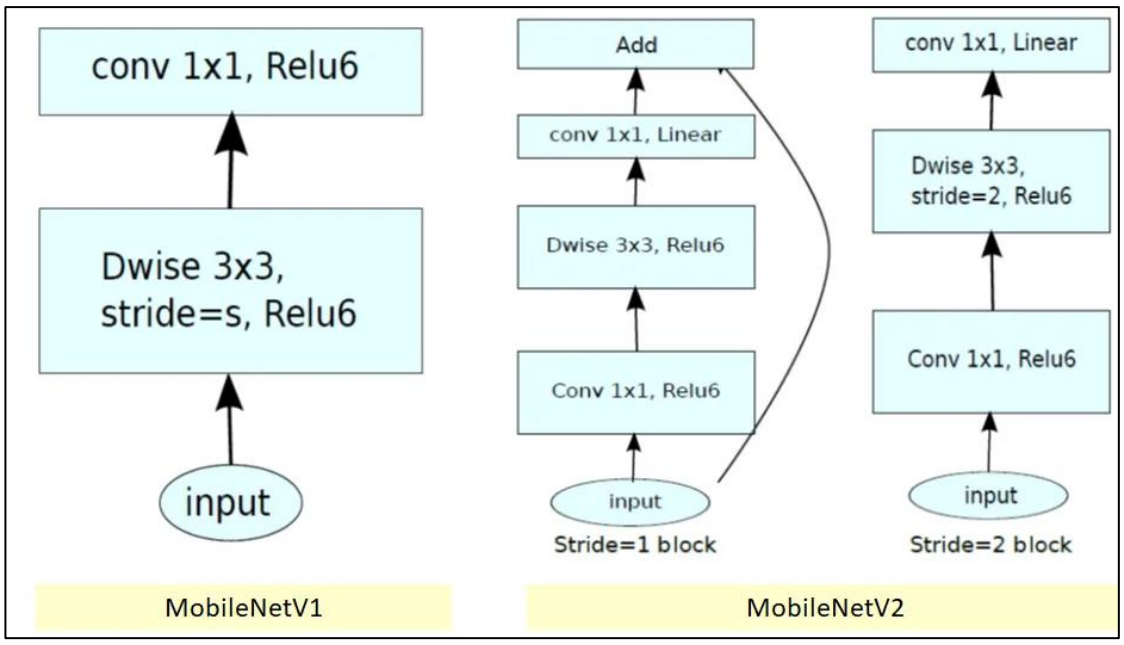

Figure 2.2 MobileNetV1 (i.e., MobileNet) and MobileNetV2 Comparisons

Source: (Sandler et al., 2018)

Moreover, it shows that the MobileNetV2 (Sandler et al., 2018) introduces the inverted residual block for feature extraction and another block to perform spatial down-sampling. More importantly, the inverted residual structure with a linear activation helps prevent information loss that can occur due to non-linearity during data compression. On that note, Figure 2.3 shows the modifications in MobileNetV3 (Howard et al., 2019) based on MobileNetV2, where it integrates the Squeeze-and-Excitation (SE) block and the hard-swish activation function to improve accuracy. Furthermore, MobileNetV3 was designed using hardware-aware neural architecture search techniques, including Neural Architecture Search (NAS) and Network Adaptation (NetAdapt), to optimise both accuracy and real-device latency.

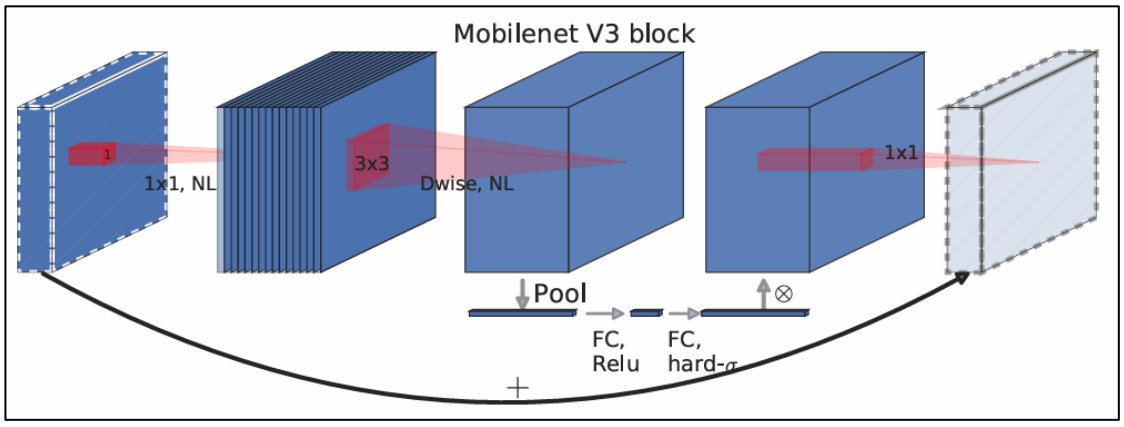

Figure 2.3 MobileNetV3 Basic Convolution Operation

Source: (Howard et al., 2019)

Besides that, the ShuffleNet architecture was proposed to improve computational efficiency for edge applications by introducing group convolution, depthwise convolution, and channel shuffle operations. The key idea is to reduce computational complexity by partitioning input channels into multiple groups and using channel shuffle to enable cross-group feature interaction. Figures 2.4(a) and 2.4(b) illustrate the basic convolution block and the spatial down-sampling block of ShuffleNetV1, respectively. Consequently, the extensive use of group convolution and depthwise convolution in ShuffleNetV1 led to increased memory access cost and data fragmentation. For that, ShuffleNetV2 (Ma et al., 2018) removes heavy group convolution usage, introduces channel split, and relocates channel shuffle after concatenation in order to reduce the issues while preserving the efficient concept of

parallel feature extraction. Figures 2.4(c) and 2.4(d) present the basic convolution block and spatial down-sampling block of ShuffleNetV2, which are refinements of 2.4(a) and 2.4(b), respectively.

Last but not least, the VGG-16 (Simonyan & Zisserman, 2014) architecture was one of the early deep CNN models developed to investigate the effect of network depth on large-scale image recognition tasks. It primarily focuses on improving classification accuracy by increasing the number of convolutional layers through the consistent use of small (i.e., 3 × 3) convolutional filters, while maintaining a simple and conventional CNN design, as shown in Figure 2.5. Despite its disadvantages, particularly the large number of parameters, its strong feature extraction capability and straightforward architectural design have enabled VGG-16 to become an important benchmark and a novel architecture. By far, it has significantly influenced many CNN architectural developments, modifications, and optimisations, which include the lightweight CNN designs.

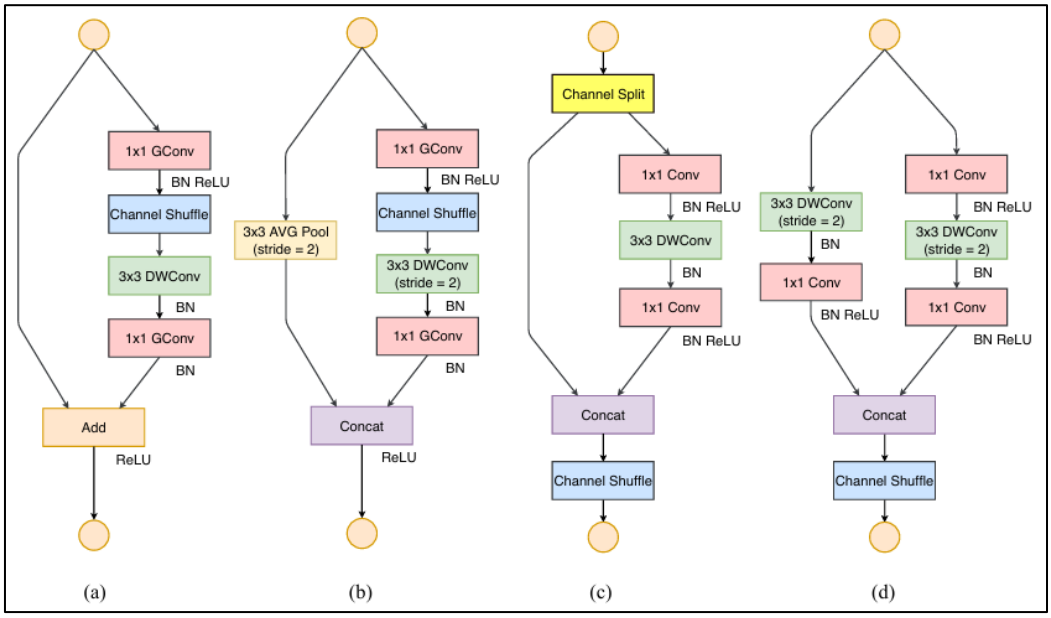

Figure 2.4 ShuffleNetV1 (i.e., (a) and (b)) and ShuffleNetV2 (i.e., (c) and (d)) Comparisons

Source: (Ma et al., 2018)

In short, the differences of lightweight CNN models have different advantages for edge deployment. For instance, MobileNetV2 reduces computation using depth-wise separable convolution and inverted residual blocks, while MobileNetV3 adds

additional features such as Squeeze-and-Excitation and hard-swish activation. On the other hand, ShuffleNetV2 focuses on reducing both computation and memory access cost through channel split, pointwise convolution, and channel shuffle operations. In order to evaluate the reliability of these models, results from different training methods (i.e., transfer learning and without transfer learning) can helps to justify the best-performed model. In conjunction, the main focus on classification tasks requires lower computational resources compared with regression tasks. Therefore, optimization technique should be designed and usually differ according to the models, training methods, classification or regression tasks, and target edge device to achieve better performance.

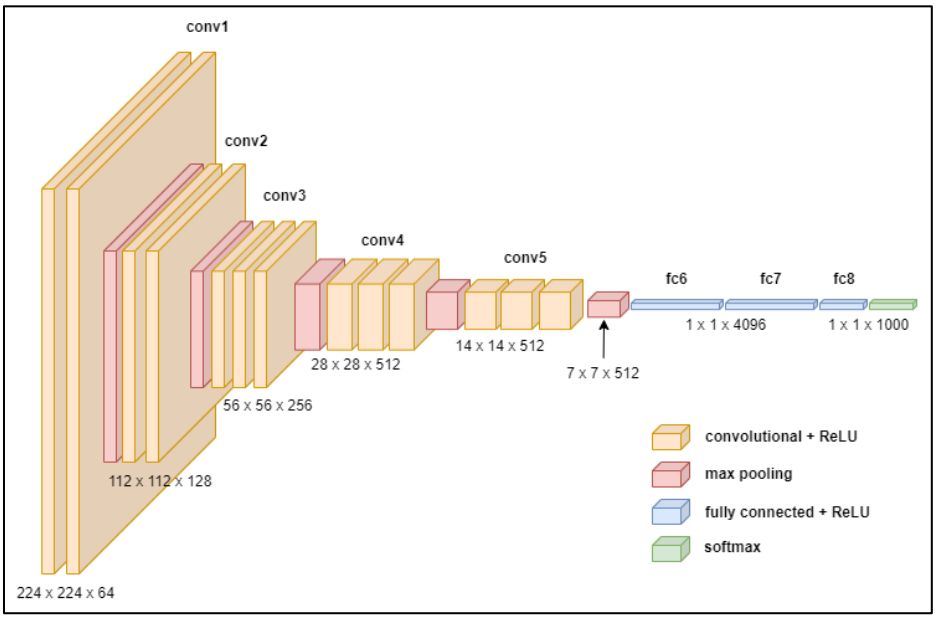

Figure 2.5 Basic VGG-16 Architecture

2.3 Types of CNN

Next, as for the types of lightweight CNN architectures in Table 2.1, most of the research used for classification, with a few of them used for regression, and some combined both classification and regression. Based on Figure 2.6, it suggests that the classification type was the main approach that was being researched, with 34 (91.9%) and the regression type, with 1 (2.7%), was much more complicated to be researched on. Since the regression types of CNN require continuous data for prediction, its implementation for resource-constrained devices may require a higher computational usage compared to the classification types, and thus, the result indicated that only minor

research has been done for regressions task. However, there were still a few studies that applied the lightweight regression CNN, and some also used it in hybrid models with classification and regression, with 2 (5.4%). In short, most of the research in lightweight CNN was leaning towards the classification types of CNN.

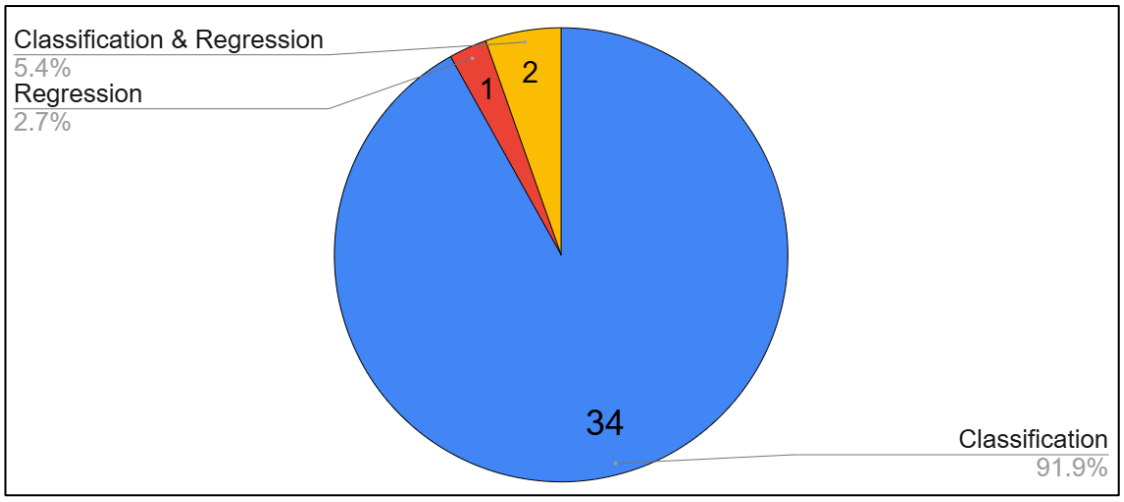

Figure 2.6 The Common Types of CNNs

2.4 Models Application

From the aspect of applications, Table 2.1 depicts that this field of research is very comprehensive. The studies were conducted from various fields including industrial manufacturing (MFG) with 8 (21.6%), public surveillance and safety with 8 (21.6%), health with 4 (10.8%), waste management with 2 (5.4%), agriculture (i.e., animal and plant) disease detection with 2 (5.4%), and military with 1 (2.7%) as illustrate in Figure 2.7. Aside from that, the highest number of applications were for general-purpose (GP) uses, with 12 (32.4%), which shows the flexibility and reliability of the network to be used with many different applications.

2.5 Type of Edge Devices

On the other hand, based on Table 2.2, 20 studies were filtered out for the performance analysis of a lightweight CNN architecture deployed in an edge device. Aside from these 20 research studies, the other 17 research studies earlier did not continue using edge devices, while only evaluating their lightweight CNN with some of the benchmark lightweight models. Figure 2.8 summarises the types and numbers of

edge devices based on the research in Table 2.2. Firstly, it can be seen that edge devices can be categorised into two types, which are on-the-shelf devices, including several system-on-a-chip (SoC) (e.g., NVIDIA Jetson series, Raspberry Pi series, etc.), microcontroller (MCU) with its development board, such as the NUCLEO-F767ZI with STM32H743, and Android smartphones. Meanwhile, the off-the-shelf devices include those being implemented as Field-Programmable Gate Arrays (FPGA) (e.g., Xilinx Zynq 7Z100, ZYNQ Z7-Lite 7020, and Intel Arria 10 GX1150).

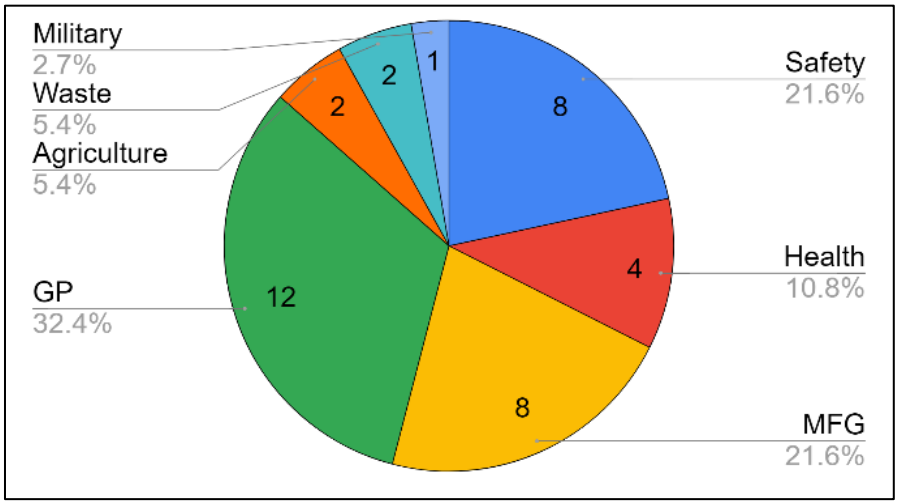

Figure 2.7 Various Applications of Lightweight CNN Use for Different Sectors of Industry

Hence, most of the recent study shows that NVIDIA Jetson Nano as the most used edge device with 7 (31.8%), followed by Raspberry Pi 4 with 3 (13.6%), and FPGA also with 3 (13.6%) while Others (SoC) with 7 (31.8%) represent the other types of SoC edge devices that were used only once. In a nutshell, the NVIDIA Jetson series and Raspberry Pi are the most commonly used edge devices in lightweight CNN research due to their accessibility, developer support, and balance of cost and performance. Typically, the Jetson series provides GPU acceleration for AI workloads, while Raspberry Pi offers a low-cost CPU-based alternative suitable for applications where GPU availability is limited.

2.6 Evaluation Types and Metrices

Last but not least, for the evaluation types and metrices, most of these studies focused on accuracy, time, speed, sensitivity, model size, computation complexity, and

others. Moreover, several metrices, such as accuracy, recall, precision, F1-score, inference time, speed, latency, parameters, floating point operations (FLOPs), and others, were always being used to ensure that the lightweight CNN for edge devices' performance was optimised. Some of the evaluation metrices represent each of the evaluation types, as the metrices are the specific assessment of each of the evaluation types. For example, Accuracy and Top-1 Accuracy are the evaluation metrices for the evaluation type of accuracy, Inference Time and Latency are the evaluation metrices for the evaluation type of time, and FLOPs and Multiply-Accumulate Operations (MACs) are the evaluation metrices for the evaluation type of computation complexity. Table 2.2 describes some of the common evaluation types and metrices that were used in those studies. Furthermore, Figure 2.9(a) and Figure 2.9(b) pointed out the number of common evaluation types and metrices. Generally, accuracy with 15 (29.4%) and 12 (17.6%) was the most used evaluation type and metric, respectively, based on recent studies.

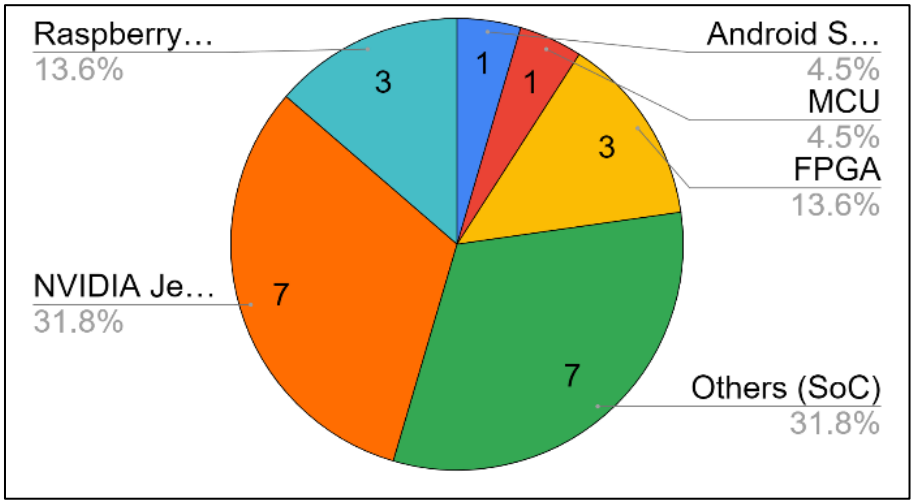

Figure 2.8 Some of the General Edge Devices Used to Embed a Lightweight CNN Architecture

2.7 Performance Comparison

Finally, the performance of all the lightweight CNNs that were deployed in edge devices was compared in terms of its respective average accuracy as one of the typical evaluation parameters and summarised in Table 2.2. Despite that, there were still a few studies that did not evaluate the accuracy of the model on an edge device while focusing more on the precision, time, and others. Figure 2.10 shows the illustration of each edge

device's performance in average accuracy. While most of them have an average accuracy of around 95% and above, a few of them only have an average accuracy of around 75%. This is because taking into account many factors, such as different specifications of each edge device and the input database of the lightweight CNN used in their respective applications, influenced the performance comparison. As such, to compare each of those edge devices' performance is not relevant, however it can still be used for expected performance in terms of average accuracy threshold value for future research of a lightweight CNN in a resource-constrained device.

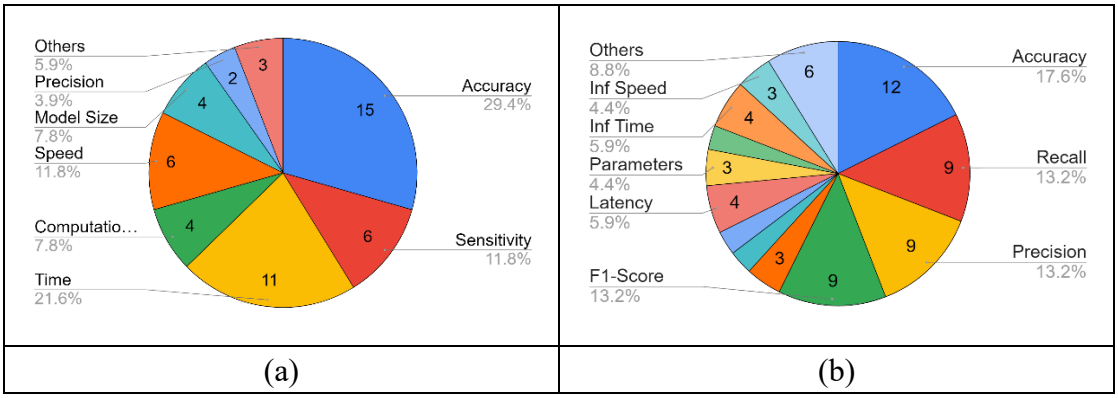

Figure 2.9 (a) The Evaluation Types and (b) the Evaluation Metrices Used Frequently in the Recent Studies.

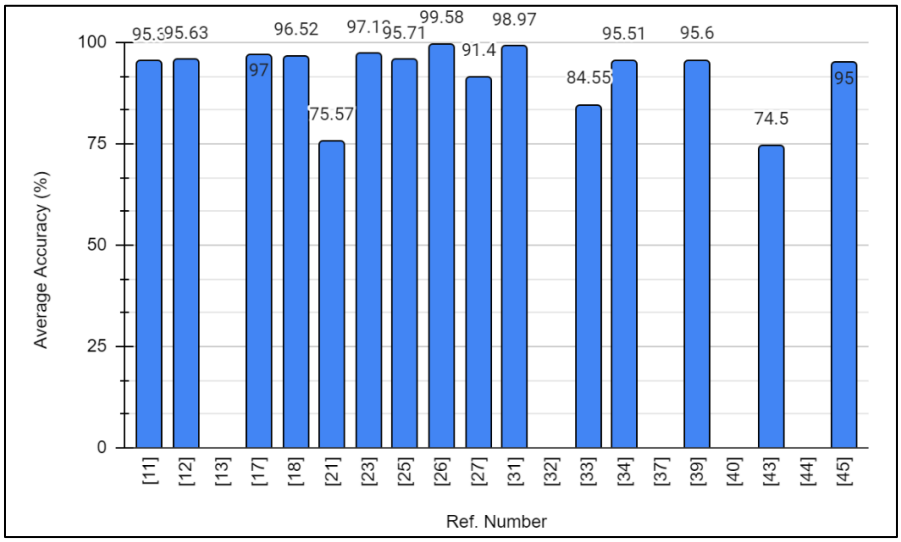

Figure 2.10 Performance Comparison in terms of Average Accuracy for Each Edge Device

2.8 Summary

In conclusion, the literature of this study was written in order to understand the current trend of lightweight convolutional neural network architecture used in edge devices. With that, several lightweight CNN architectures and models were identified, including the types of CNN, their applications, edge devices used, evaluation types and metrices, as well as a simple performance comparison regarding their average accuracy. Despite the growing popularity of edge AI, several research gaps, such as the lack of fair comparison and generalisation to evaluate lightweight CNN models' performances and the benchmark to find out the best-performing model, are still in the grey zone. Also, the inconsistent performance for the same model when applied to different applications indicates the need for further optimisation and improvement of lightweight CNN models. One of the main factors for these issues is related to the difficulty of balancing the trade-off between the inference accuracy, latency, and computation capacity of the model, and thus, most of the recent lightweight models remain application-specific models. As the scope of this field of study is still in the early stage, many contributions and novel approaches are required to ensure comprehensive advancements of edge device technology with the utilisation of lightweight CNNs. Thus, with the findings that have been obtained here, in the next chapter, the research methodology of this study will be constructed as one of the approaches to further closing the research gap for edge computing applications.

Table 2.1
Research Database Literature Classifications

Ref. Num.	Architectures / Models	Types of CNN	Applications
(Bhagat et al., 2024)	Lite-MDC	Classification Types of Pigeon Pea's Diseases	Plant Disease Detection for Pigeon Pea
(Chang & Zheng, 2024)	VGG-16	Classification of Cracked / Non-Cracked Surfaces	Automated Crack Detection in Building Inspection and Maintenance
(Neogi et al., 2024)	SDLM	Classification of Cataract / Non-Cataract Eye	Cataract Eye Detection
(Thakuria & Erkinbaev, 2024)	ShuffleNetV2_YOLOv5s	Classification Types of Canola Kernel Grades	Real-Time Canola Damage Detection
(Thwal et al., 2024)	OnDev-LCT	Classification	General Purpose for Image Classification

(Tran et al., 2024)	Zero-FVeinNet	Classification Types of Fingers Vein	Finger Vein Recognition
(Xiao et al., 2024)	YOLOv5	Classification of the Maturity of Blueberry Fruit	Blueberry Fruit Maturity Detection
(Bonam et al., 2023)	- MobileNetV2 - CondenseNetV2 - ShuffleNetV2	Classification of Fabric, Surface, and Casting Defect	Product Defect Detection in MFG Industries
(Burra et al., 2023)	MobileNetV2	Classification Types of Fabric Defect	Fabric Defect Detection in Textile MFG
(Gang et al., 2023)	CCNNet	Classification Types of TCM	Traditional Chinese Medicine Image Classification
(Kim et al., 2023)	MobileNet	Classification	General Purpose for Image Classification
(Kong et al., 2023)	- VGG16_BN - ResNet-50 - RegNet-X	Classification	General Purpose for Image Classification
(S. Li et al., 2023)	CH-CNN	Classification	General Purpose for Image Classification
(X. Li et al., 2023)	MobileViTFace	Classification of Different Breeds of Sheep	Sheep Face Recognition
(Y. Liu et al., 2023)	- VGGNet-16 - ResNet-50/56/110 - GoogLeNet - DenseNet-40	Classification	General Purpose for Image Classification
(Luo et al., 2023)	LiteCNN	Classification of Different Types of Plant Diseases	Plant Disease Identification
(Narasimha Prasad et al., 2023)	ShuffleNetv2	Classification of Defective / Non-defective Casting	Casting Defect Detection
(Rani et al., 2023)	CondenseNetV2	Classification Types of Surface Defect	Surface Defect Detection in Industrial Intelligent Production
(Shehada et al., 2023)	CNN	Classification Types of Facial Emotion	Facial Emotion Recognition for VIP
(Shi et al., 2023)	EBNAS	Classification	General Purpose for Image Classification
(Xie et al., 2023)	YOLOv5s-BiFPN	- Classification of Pig Body Region of Interest (ROI) - Regression of Pig Body Temperature	Pig Body Temperature Automatic Detection for Early Disease Warning
(Almeida et al., 2022)	EdgeFireSmoke	Classify the Occurrence of Forest Fires	Fire-Smoke Detection of Forest Fires
(Ju et al., 2022)	TripleNet	Classification	General Purpose for Image Classification
(Kalgaonkar	CondenseNeXtV2	Classification	General Purpose for Image

& El-Sharkawy, 2022)			Classification
(Kristiani et al., 2022)	- InceptionV3 - MobileNet - VGG16	Classification Types of Face with Mask / without Mask	Face Mask Classification
(Lu et al., 2022)	HFENet	Classification of Defective / Non-defective Ceramic Tile Surface	Ceramic Tile Surface Defect Detection
(Song & Guo, 2022)	ShuffleNet	Classification of 3D Object Images	3D Object Recognition for 3D Scanning Technology
(S. Wang et al., 2022)	MobileNetv2	Regression of Crowd Density Estimation	Estimating Crowd Density for Public Security Management
(Xu et al., 2022)	RDPNet	Classification	General Purpose for Image Classification
(He et al., 2021)	Ensemble Binarized DroNet (EBDN)	- Classification Task for Collision-Avoidance - Regression Task for Prediction of Desired Steering Angle	Autonomous Driving for UAV
(Mocerino & Calimera, 2021)	BC-Net	Classification	- General Purpose for Image Classification - Speech Recognition of Keyword Spotting - Facial Expression Recognition
(Qin et al., 2021)	MobileNet-v2	Classification Types of Solid Waste	Waste Classification for Solid Waste Management
(A. K. Sharma et al., 2021)	DenseNet	Classification	General Purpose for Image Classification
(B. Wang et al., 2021)	MobileNetV3	Classification Types of Icing Grades	Icing Monitoring of Transmission Lines
(Xia et al., 2021)	SparkNet	Classification	General Purpose for Image Classification
(Yoo et al., 2021)	MobileNetV2 & SqueezeNet	Classification Types of Waste	Reverse Vending Machine for Types of Waste Recycled
(Yu et al., 2021)	ASIR-Net	Classification Types of Different Ground Vehicle Targets	Automatic Target Recognition (ATR) in Synthetic Aperture Radar (SAR) Images for Military Surveillance

Table 2.2
Research Database Edge Devices' Performance Classifications

Ref. Num.	Edge Device Types	Evaluation Types	Evaluation Metrices	Ave. Accuracy (%)
(Chang & Zheng, 2024)	Raspberry Pi 3B+	- Accuracy - Model Size - Robustness	- Accuracy - Recall - Precision - F1-Score	95.30
(Neogi et al., 2024)	Android Smartphone	- Accuracy - Model Size - Time	- Accuracy - Inference Time - Parameters	95.63
(Thakuria & Erkinbaev, 2024)	NVIDIA Jetson Nano	- Speed - Sensitivity	- Precision - Recall - F1 Score - Inference Speed	-
(Bonam et al., 2023)	NVIDIA Jetson Nano	- Accuracy - Sensitivity - Specificity	- Accuracy - Recall - Precision - F1-score	97.00
(Burra et al., 2023)	NVIDIA Jetson Nano	- Accuracy - Sensitivity	- Accuracy - Recall - Precision - F1-Score	96.52
(Kong et al., 2023)	- NVIDIA AGX Xavier - NVIDIA Jetson Nano	- Accuracy - Computations Complexity - Time - Model Size	- Top-1 Accuracy - MACs - Latency - Parameters	75.57
(X. Li et al., 2023)	NVIDIA Jetson Nano	- Accuracy	- Accuracy - Precision - Recall	97.13
(Luo et al., 2023)	ZYNQ Z7-Lite 7020 FPGA	- Accuracy - Speed - Time	- Accuracy - Inference Speed - Latency	95.71
(Narasimha Prasad et al., 2023)	NVIDIA Jetson Nano	- Accuracy - Sensitivity	- Precision - Recall - F1-score - Accuracy	99.58
(Rani et al., 2023)	NVIDIA Jetson Xavier Nx	- Accuracy - Sensitivity	- Accuracy - Recall - Precision - F1-Score	91.40
(Almeida et	NVIDIA Jetson Nano	- Accuracy	- Accuracy	98.97

al., 2022)		- Sensitivity - Time	- Recall - Precision - F1-Score - Hamming Loss	
(Ju et al., 2022)	Raspberry Pi 4	- Time - Computations Complexity	- Latency - FLOPS	-
(Kalgaonkar & El-Sharkawy, 2022)	NXP BlueBox 2.0	- Model Size - Accuracy - Time - Computations Complexity	- FLOPS - Parameters - Top-1 Accuracy - Inference Time	84.55
(Kristiani et al., 2022)	Raspberry Pi 4	- Accuracy - Speed - Loss	- Accuracy - Precision - Recall - F1-Score	95.51
(S. Wang et al., 2022)	Raspberry Pi 4	- Speed - Computations Complexity - Time	- Mean Absolute Error (MAE) - RMSE - Inference Speed and Time - FLOPS	-
(He et al., 2021)	Xilinx Zynq 7Z100 FPGA	- Accuracy - Precision - Speed	- Root Mean Square Error (RMSE) - F1-Score - FPS	95.60
(Mocerino & Calimera, 2021)	NUCLEO-F767ZI with STM32H743	- Time	- Mean Square Error (MSE) - Latency	-
(B. Wang et al., 2021)	Huawei Atlas 200 DK	- Accuracy - Precision - Time	- Accuracy - Time - FPS	74.50
(Xia et al., 2021)	Intel Arria 10 GX1150 FPGA	- Time - Speed	- Inference Time - Speed Up	-
(Yoo et al., 2021)	- NVIDIA Jetson Nano - NVIDIA Jetson TX1	- Accuracy - Time	- Accuracy	95.00

CHAPTER 3
RESEARCH METHODOLOGY

3.1 Introduction

This research methodology describes a systematic strategy for constructing and optimising a lightweight CNN model for image classification tasks on an edge device. The key purpose is to ensure that the optimised model achieves high accuracy while retaining computing efficiency appropriate for edge devices. Generally, the methodology includes a thorough analysis of the strategies concerning the model selection, optimisation techniques, and performance evaluation metrics to fulfil the research objectives.

3.2 Research Methodology Overview

Based on Figure 3.1, the research approach follows a defined procedure, as seen in the flowchart. The procedure contains several stages representing each of the research objectives. Primarily, the first stage of the research procedures begins with Stage 1, which is to investigate the current lightweight CNN models that are best for the image classification task on an edge device. This investigation is conducted in order to identify the best-performing model that will be utilised in this research.

After that, Stage 2 will be performed, which is to improve and optimise the performance of the identified lightweight CNN in terms of several appropriate evaluation metrics. This stage entails modifying the structure of the identified lightweight CNN architecture using several optimisation techniques to improve its performance. After that, the optimised model will be executed, and its performance will be justified on external hardware (i.e., a desktop computer).

Then, Stage 3 represents Research Objective 3, which is to evaluate the performance of the optimised lightweight CNN model using classification accuracy and inference time as the core evaluation metrics. For this, the optimised model trained on several identified image classification datasets will be deployed on edge devices, where a comparison study will be done to determine the model's overall performance.

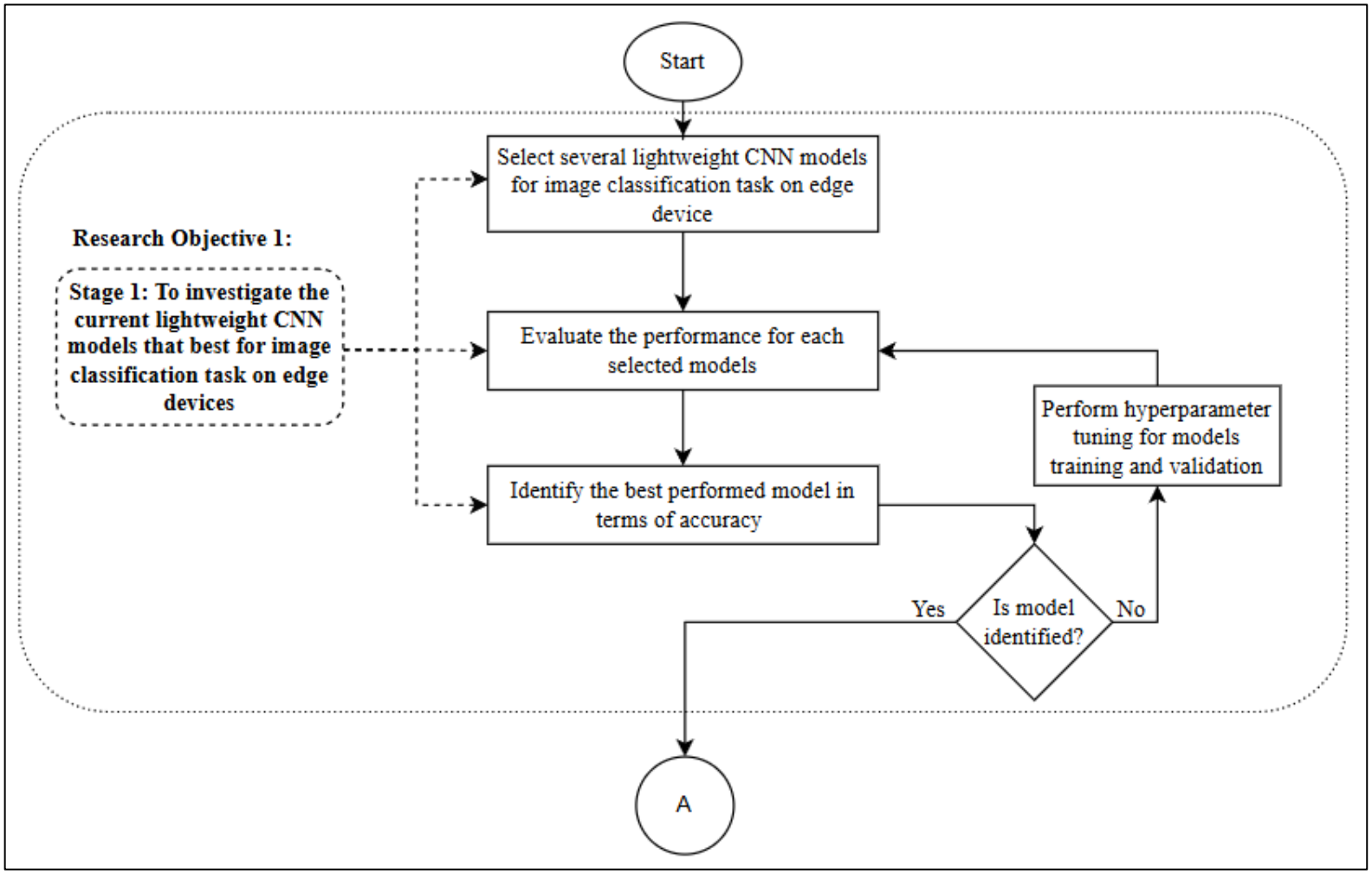

Figure 3.1 (a) Flowchart of this Research Methodology Correspond to Research Objective 1

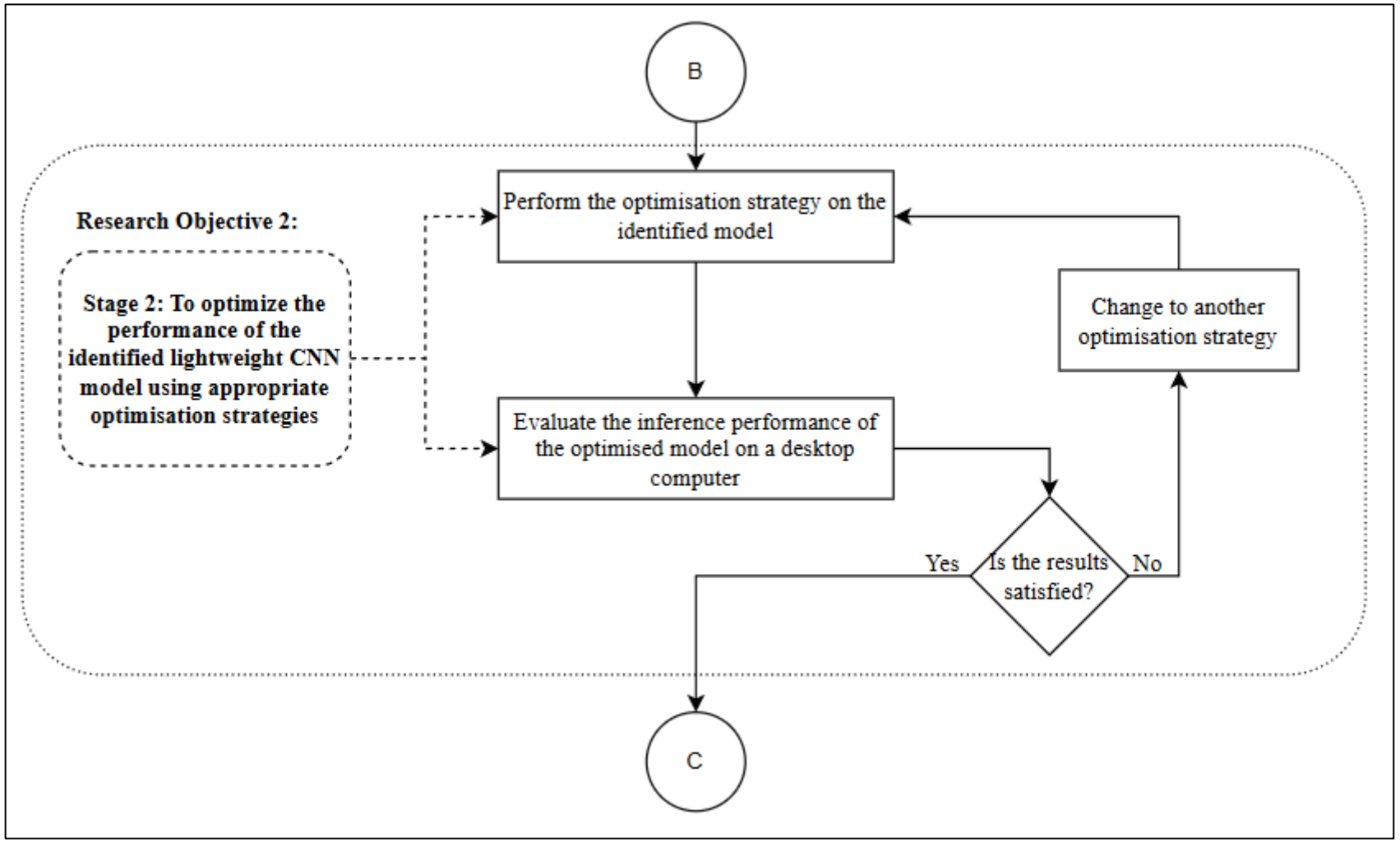

Figure 3.1 (b) Flowchart of this Research Methodology Correspond to Research Objective 2

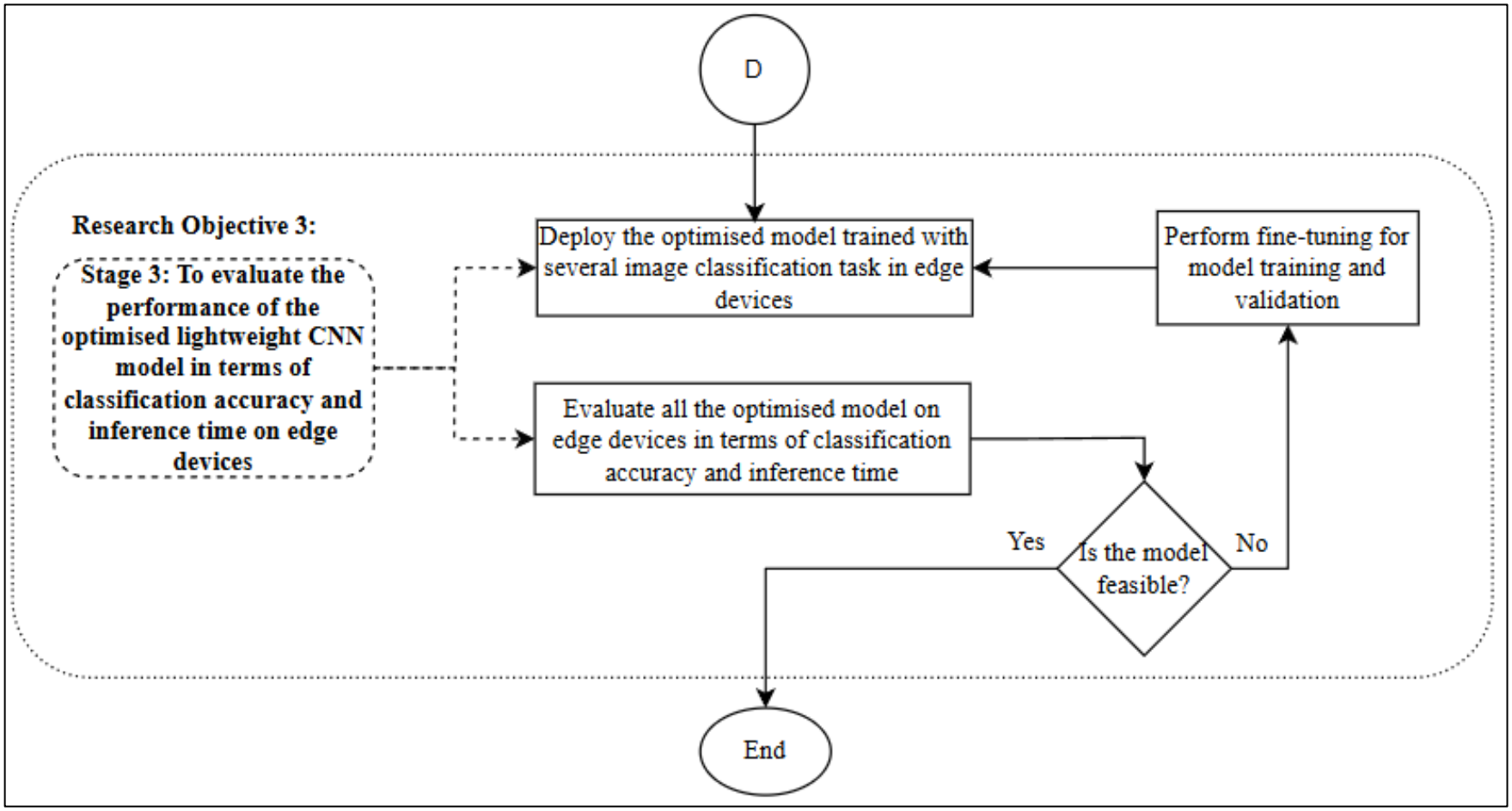

Figure 3.1 (c) Flowchart of this Research Methodology Correspond to Research Objective 3

In short, this study employs an experimental research work, which entails studying current CNN models, selecting and identifying the best-performing model for Stage 1, optimising the identified model in Stage 2, and deploying it on an edge device, including assessing its performance with defined criteria for Stage 3.

3.3 Stage 1: Investigate the Current Lightweight CNN Models that Best for Image Classification Task on Edge Device

This stage will explain a thorough process of investigating several lightweight CNN models and sequentially identifying the best model for Stage 2. First, based on the previous literature review, this process starts with the selection of commonly used models from other researchers. After that, those selected models will be evaluated accordingly in order to identify the best-performing model. In short, this stage is taken into consideration and conducted to achieve the first objective of this research.

3.3.1 Lightweight CNN Models Selection

In detail, Figure 3.2 shows the first subprocess in this stage. Initially, in order to determine the best CNN models for edge devices, a thorough literature analysis of cutting-edge lightweight CNN models is undertaken. The major emphasis is on models that are optimised for efficiency, low computational overhead, and good classification performance. Aside from that, not all models discussed in the literature have been deployed on edge devices, as some of them were still being tested on high-end and middle-end computers. For that, this research has considered some frequently used architectures as the primary selection criteria based on subsection 2.2 in the previous chapter.

One of them is MobileNet, which has been discovered to be the most used model in recent years. MobileNet is known for its balance of accuracy and computing efficiency, making it suitable for mobile and embedded applications. Moreover, ShuffleNet is also considered for this research, as it has also been moderately used by several researchers. ShuffleNet is defined by its key architectural feature, as it uses group convolutions and channel shuffling to minimise computation while preserving accuracy. Other than that, VGG16 is recognised as a selected model, for it has also been similarly used as ShuffleNet. VGG16 is known for its compact architecture, which

employs a conventional deep stacked convolutional structure to attain high accuracy with a slower learning rate.

Then, the availability of the selected models is also checked from an open-source machine learning library, such as PyTorch, to ensure that these models can be utilised and developed with a fully featured framework. For that, several variances of the model from MobileNet, such as MobileNetV2 (Sandler et al., 2018) and MobileNetV3 (Howard et al., 2019), are further selected. For MobileNetV3, there are two more variations, which are the MobileNetV3 Small model and the MobileNetV3 Large model. On the other hand, the ShuffleNetV2 (Ma et al., 2018) is chosen instead of ShuffleNet, with four more variations such as ShuffleNetV2x0.5, ShuffleNetV2x1.0, ShuffleNetV2x1.5, and ShuffleNetV2x2.0. With that, including VGG16 (Simonyan & Zisserman, 2014), the total number of the selected models for further analysis is 8.

3.3.2 Performance Evaluation for Selected Models

For this part, the selected models from the previous subprocess will be experimented with and evaluated as depicted in detail, as shown in Figure 3.3. First, there are two sets of experiments that were conducted, the first one is to conduct a transfer learning as preliminary work for this research. Meanwhile, the second set will experiment with the selected models without transfer learning, meaning that the selected models will be trained from scratch using randomly initialised weights.

Here, in both sets of experiments, primarily, an Anaconda environment is set up on a desktop computer in order to manage the necessary libraries' dependencies for the training and validation process. As such, for the model framework, PyTorch is mainly used with regard to both sets of experiments. In the first set of experiments, which is for transfer learning, the obtained pre-trained models are based on PyTorch models that were trained using the ImageNet dataset, whereas in the second set of experiments, which is without transfer learning, the same architectures from PyTorch are used without the pre-trained models, as the selected models are trained and validated from scratch.

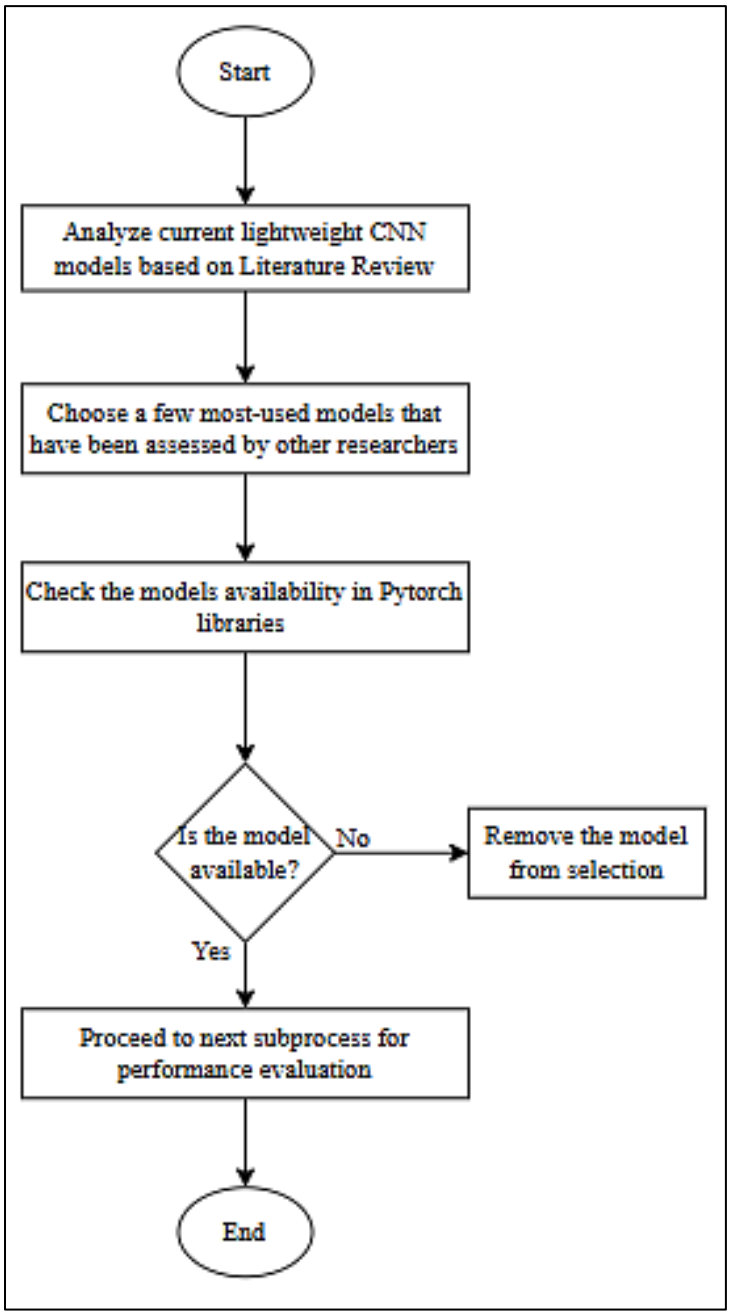

Figure 3.2 Subprocess of Lightweight CNN Models Selection

Moreover, for the image classification dataset as shown in Table 3.1, a fixed dataset is used on all the selected for training and validation. Specifically, this dataset is a dataset of image classification for different classes or types of leaf diseases for a groundnut plant (Manvikar & Reddy, 2023). The classes of each type of leaf disease are divided into 6 classes, such as early leaf spot, 1065 images of early rust, 1491 images of late leaf spot, 1255 images of nutrition deficiency, 1315 images of rust, and 1462 images of the healthy leaf class. All in all, the total number of sample images for training is 7910 for all classes combined. For data augmentation, 30% of each class's

sample images are used for validation. Furthermore, all of the images for both training and validation are being randomly organised in different directions and with different backgrounds. However, image size and channel remain fixed for training and validation, which are 256x256x3.

On top of that, the groundnut plant leaf disease dataset was selected as the primary dataset for this stage because it satisfied several important criteria for evaluating lightweight CNN architectures. First, it is a well-structured multi-class dataset comprising six distinct disease categories with a relatively balanced distribution of images, making it suitable for a fair comparison of different classification models. Second, the dataset represents a real-world agricultural disease detection application, which aligns with the growing adoption of edge AI for smart agriculture. Third, its moderate level of complexity provides a realistic benchmark for assessing model performance, as it is neither overly simplistic nor excessively complex. Furthermore, the dataset is publicly available, ensuring the reproducibility and transparency of the experimental results. Finally, the findings obtained from this dataset can be readily extended to similar image classification tasks in agriculture and other edge AI applications, making it an appropriate benchmark for the evaluation of lightweight CNN models.

Table 3.1
Groundnut Plant Leaf Disease Dataset Specifications

Name of the dataset	Groundnut plant leaf diseases (Manvikar & Reddy, 2023)
Input size	256x256x3
Number of classes	6
Total images for training and validation	7910
Total images for inference	2451

After that, the other settings for software tools and desktop computer hardware are configured. For that, Python is installed as the programming language and Compute Unified Device Architecture (CUDA) Application Programming Interface (API) is also installed to enable the NVIDIA GeForce GTX 1060 6GB GPU as the computing tool for the model training and validation. Also, this system is executed on a Windows 11 Pro Operating System (OS) with an Intel i5-3570K CPU and 12 GB RAM on a desktop

computer platform. Table 3.2 shows the details of the desktop computer platform specifications.

Table 3.2
Desktop Computer Platform Configurations

OS Platform	Windows 11 Pro
CPU	Intel i5-3570K CPU
RAM	12 GB
GPU	NVIDIA GeForce GTX 1060 6GB
Environment	Anaconda
Framework	PyTorch

Last but not least, hyperparameter settings for each of the selected models are also configured as the final step before executing each of the training and validation. In order to obtain more reliable comparisons for this analysis, most of the hyperparameter settings for the models are kept at default and fixed values to minimise their variables. However, except for VGG16, its learning rate has to be much smaller than the others in order to avoid loss function break and keep its weight stabilised due to the absence of normalisation in its architecture (Simonyan & Zisserman, 2014). Also, the optimizer for all these model is based on Stochastic Gradient Descent (SGD) algorithm. The training approach consisted of several hyperparameters, as shown in Table 3.3.

Table 3.3
Training Hyperparameter for the Selected Model

Models	Epochs	Batch Size	Learning Rate	Momentum	Weight Decay
MobileNetV2	35	8	0.1	0.9	1e-4
MobileNetV3-Small	35	8	0.1	0.9	1e-4
MobileNetV3-Large	35	8	0.1	0.9	1e-4
ShuffleNetV2x0.5	35	8	0.1	0.9	1e-4
ShuffleNetV2x1.0	35	8	0.1	0.9	1e-4
ShuffleNetV2x1.5	35	8	0.1	0.9	1e-4
ShuffleNetV2x2.0	35	8	0.1	0.9	1e-4
VGG-16	35	8	0.005	0.9	1e-4

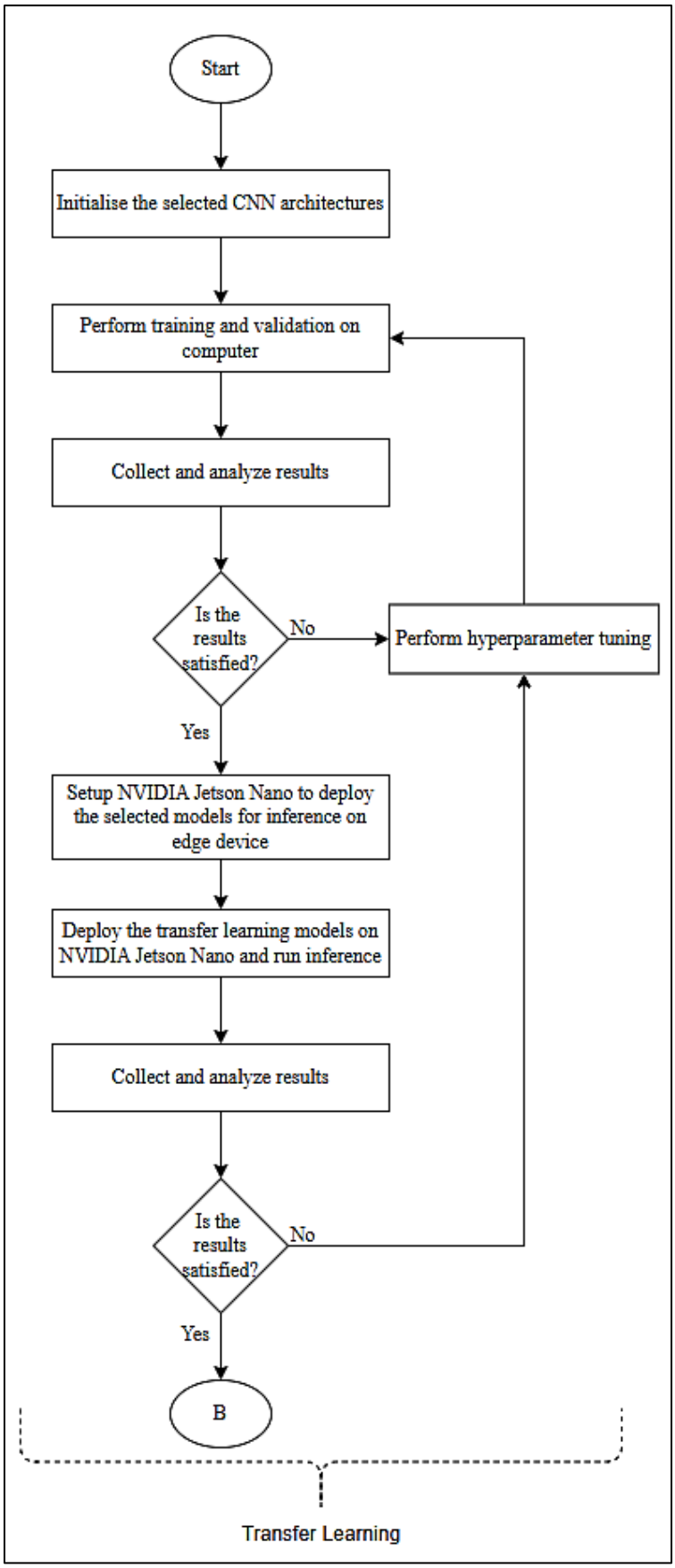

Figure 3.3 (a) Subprocess for Selected Models' Performance Evaluation with Transfer Learning

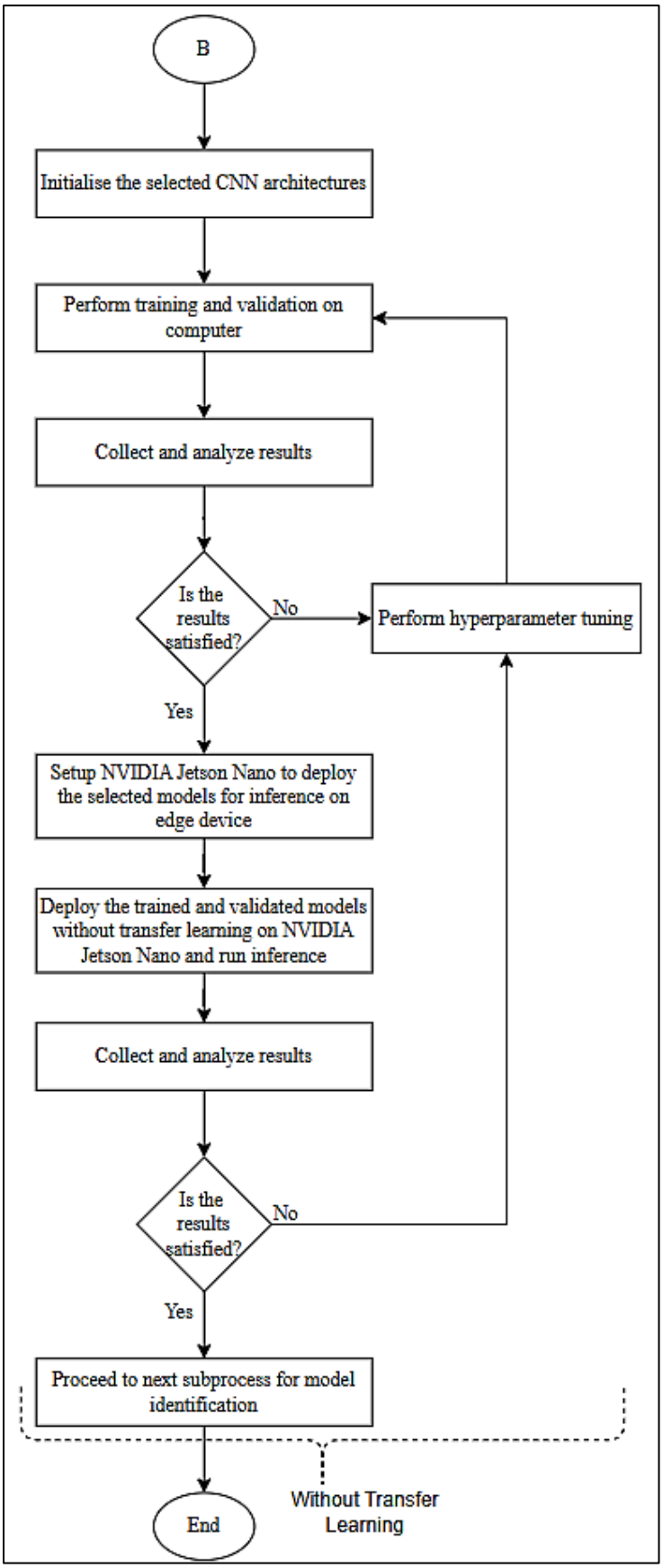

Figure 3.3 (b) Subprocess for Selected Models' Performance Evaluation without Transfer Learning

After completing the training and validation on the desktop computer, the setup on the edge device is configured to deploy and run inference as depicted in Table 3.4. For that, first, the NVIDIA Jetson Nano is selected as the edge computing platform. This device comes with Jetpack Software Development Kit (SDK) as its firmware, which includes all the necessary tools for AI development, such as CUDA, TensorRT, Open Neural Network Exchange (ONNX), and others. Also, Linux Ubuntu is used as the OS for this device, and the application environment for this work is configured within a Docker container. Other than that, Python is also used as a programming language, including the inference script.

Table 3.4
NVIDIA Jetson Nano Platform Configurations

Device	NVIDIA Jetson Nano
OS Platform	Linux Ubuntu
GPU	128-core Maxwell GPU
Environment	JetPack SDK
Framework	PyTorch

3.3.3 The Best Performed Model Identification

After having conducted both sets of experiments in the previous subprocess of this stage, here is the strategy for identifying the best-performing model, as pictured in Figure 3.4. Primarily, this subprocess starts by analysing the inferences' performance for each of the same model architectures between the one that was conducted with a pre-trained model for transfer learning and the one without transfer learning that was trained from scratch.

Mainly, the first comparison will observe the classification accuracy for each of those selected models in order to identify the best-performing model. In detail, for testing or inference, classification accuracy is basically the overall accuracy of the model. By taking the total number of correct predictions divided by the total number of sample images and multiplying by 100, the percentage accuracy of the models is obtained, as shown in (3.1).

$$\text{Accuracy (\%)} = \left(\frac{\sum Correct\ Predictions}{\sum Sample\ Images}\right) \times 100 \tag{3.1}$$

However, if the previous condition is not met, then the comparison will continue to observe the accuracy and loss taken from the training and validation of the selected models. In depth, accuracy is the most basic evaluation metric in machine learning for both training and testing. Unlike accuracy, loss is often used only for training and validation. In short, accuracy is calculated simply by taking the number of correct predictions over the total number of samples. However, loss is the summation of errors of the loss function per epoch from the training and validation of the model. Ideally, accuracy should keep increasing, while loss keeps decreasing as the training goes on.

To surmise, the identification of the best-performed model is evaluated based on the classification accuracy of its inference and followed by the accuracy and loss of its training. If both of the same model that are pre-trained with transfer learning and trained without transfer learning do not achieve the highest classification accuracy, then the best-performing model will be justified based on the stability and balanced performance throughout the training and validation process.

3.4 Stage 2: Optimise the Performance of the Identified Lightweight CNN Model using Appropriate Optimisation Strategies

In this stage, after the best-performing model has been identified in the previous stage, the identified lightweight CNN model will undergo a comprehensive improvement and optimisation process to enhance its accuracy, efficiency, and generalisation while maintaining minimal computational costs. Recently, there have been several architecture optimisation strategies introduced by other researchers. With that, the process here begins with performing an optimisation strategy on the identified model. Then, the optimised model will be validated using several different keys of evaluation metrics that will be identified. Also, the performance of the optimised model will be thoroughly evaluated in order to observe whether the model can be deployed on an edge device or if it still needs further improvement and optimisation. As such, the evaluation will be done in a desktop computer environment first in order to prove the advantages of the optimisation strategy before deploying the optimised model on edge devices.

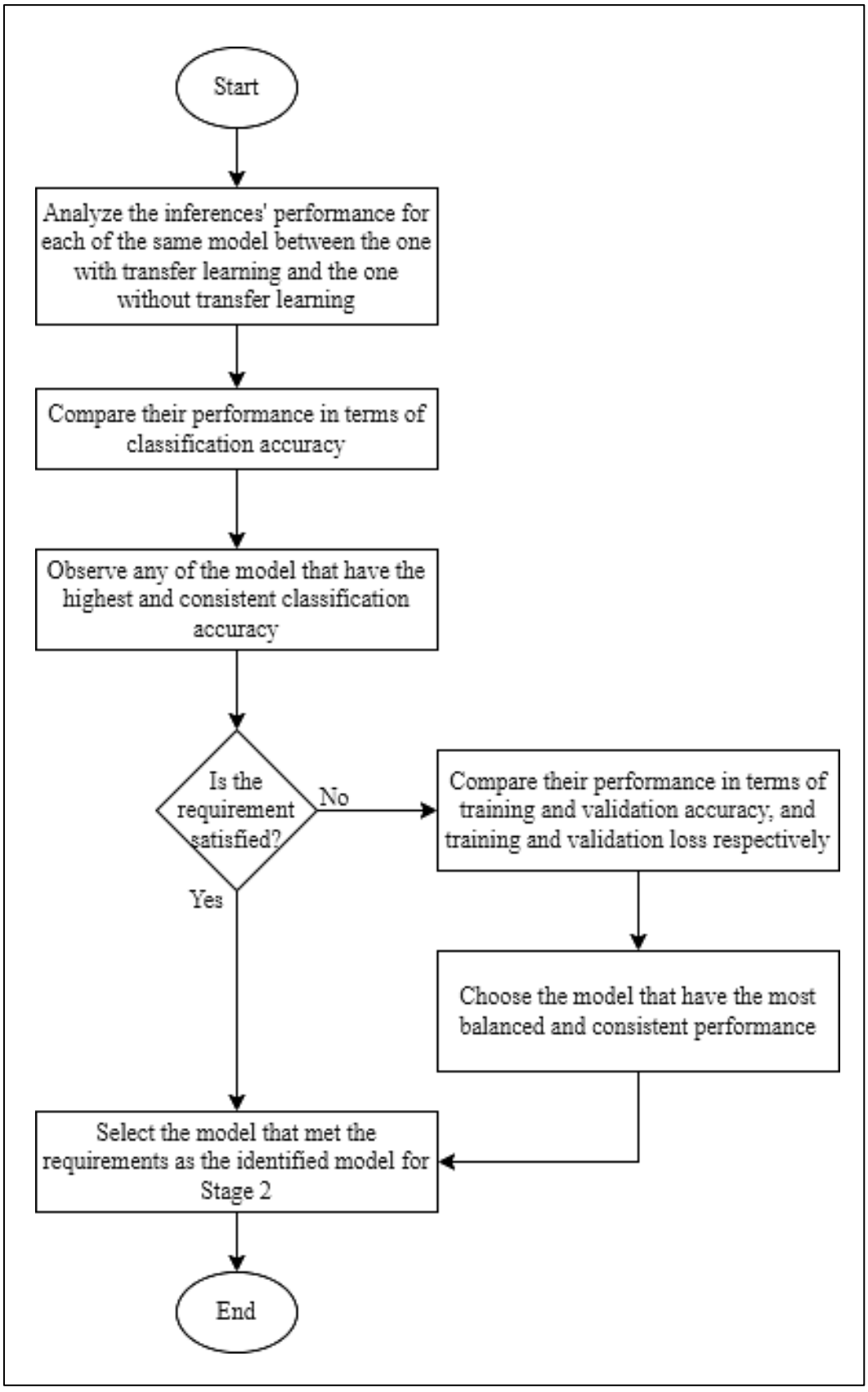

Figure 3.4 Subprocess for the Best Performed Model Identification

3.4.1 Lightweight CNN Architecture Optimisation Strategies

First of all, optimising the identified lightweight CNN model involves multiple advanced strategies aimed at improving its efficiency and accuracy while ensuring it remains computationally lightweight. Figure 3.5 summarises the subprocess of this stage. In recent years, there have been many architecture optimisation strategies introduced by other researchers to improve the performance of lightweight CNN models. Thus, in order to apply any of the strategies or techniques to the identified model, it is crucial to understand the basic concept and fundamentals behind several of the optimisation strategies. Specifically, the strategy that will be utilised in this research includes post-train channel pruning and network simplification.

Generally, channel pruning is a hardware-friendly, structured pruning technique that reduces the output dimension of the convolutional layer by removing specific output channels (i.e., feature maps), which substantially decreases the number of network parameters while maintaining the integrity of the network topology, thereby striking a balance between model compression and hardware compatibility. With that, redundant or less influential channels from convolutional layers will be removed, thus decreasing memory consumption and inference time while maintaining accuracy on an edge device. While theoretically channel pruning is supposed to reduce the number of parameters, the current PyTorch library only permits determining and masking those redundant or less important filters to zero.

Currently, there are two ways of calculation embedded in the channel pruning library that can be selected to determine the unnecessary filters. The first one is called the L1-norm, also known as the Manhattan norm. It calculates the relevance of each parameter by adding up the absolute values of the weights as in (3.2). L1-norm pruning is commonly used for structured channel pruning in convolutional neural networks because of its resilience to preserve accuracy. The second one is the L2-norm, also called the Euclidean norm, which is the square root of the sum of squared weight values as in (3.3). L2 norm-based pruning assesses the significance of weights or structured units such as channels. Because of the squaring process, this norm emphasises larger selective weights more than smaller ones, which makes it more suitable for unstructured pruning (i.e., weight pruning). Thus, in this strategy, the L1-norm will be employed to calculate, identify, and mask the channel that needs to be removed.

As the zero filters or channels with zero weights still have their overhead computations and need a custom technique to properly remove the masked filters manually, a technique called ‘Simplify’ (Bragagnolo & Barbano, 2022) from an open-source GitHub library will be integrated with Pytorch’s channel pruning. Technically, this technique simply works in two steps, starting from any secondary channels or filters. First, it removes the zero channel in the previous layer, and second, it removes the zero channel in the current layer. By keeping repeating this process, it manually modifies the model’s architecture, which includes reducing the neural network input and output channels to lower the computational costs of the model. Also, this strategy can be varied by manipulating the percentage of channel removal, which is also known as the sparsity level, from the original number of parameters in the identified model.

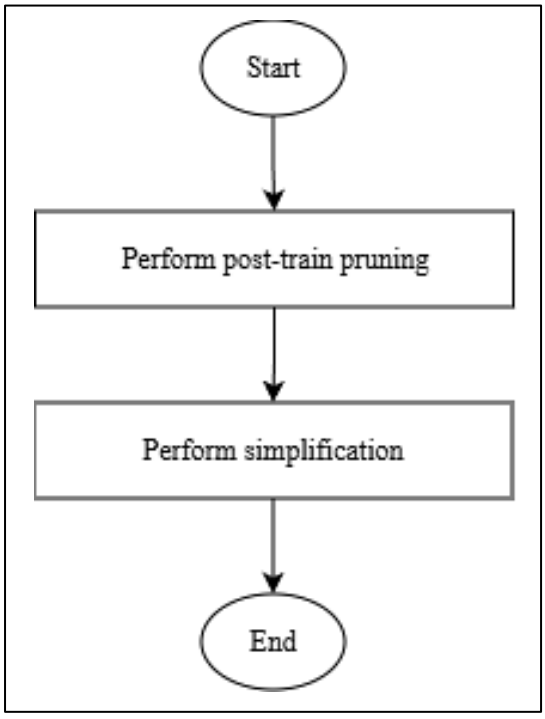

Figure 3.5 Subprocess for Optimising the Identified Lightweight CNN Architecture

In short, the combination of these two techniques will create the optimisation strategy that will be used to optimise the identified model. To make it clearer for the analysis of this research, a post-train pruning using L1-norm calculation with network simplification will be used throughout the experiment as the main optimization technique. In specific, a post-train pruning refers to the process of pruning a fully trained neural network after the training stage has been completed. Additionally, this can help to justify the feasibility and model robustness while ensuring the optimised

computational performance without sacrificing accuracy by customising the architecture accordingly on a specific sparsity level. After having applied the optimisation strategy mentioned above, the improved and optimised model will proceed to the next subprocess for evaluation using several appropriate evaluation metrics.

$$\text{L1} - \text{norm} = \| w \|_1 = \sum_{i=1}^{n} | w_i | \tag{3.2}$$

$$\text{L2} - \text{norm} = \| w \|_2 = \sqrt{\sum_{i=1}^{n} w_i^2} \tag{3.3}$$

Where:

w = weight vector of the convolutional layer

n = number of weights in the weight vector

3.4.2 Evaluate the Testing Performance of the Optimised Model on Desktop Computer Platform

For this subprocess, the testing will first be conducted on the desktop computer after the optimisation strategy is completed. For that, it is crucial to make sure that all the libraries and tools, as well as the testing or inference script, can be executed similarly on an edge device. This is to reduce variability that can interfere with the differences in computation cost between the desktop computer environments and the edge device. Moreover, further enhancements of optimisation effects on the optimised model can be clearly seen if those variabilities are avoided.

Assuming that the testing can be executed properly, the testing performance of the optimised model will be evaluated. With that, several core evaluation metrics, specifically, the number of parameters, floating-point operations, model size, classification accuracy, and inference time, will be collected and analysed. For classification accuracy, it follows the formula that has been described previously in subsection 3.2.3.

Moreover, the number of parameters is the learnable weights and biases that the model changes while training. They reflect the model's capacity to store data through all its layers and are the major cause of its memory consumption. From an assessment standpoint, the number of parameters can be determined by multiplying the number of

output channels, input channels, and kernel size (i.e., the width and height of the filter) and adding bias as shown in (3.4).

$$\text{Number of parameters} = (C_{out} * C_{in} * K_w * K_h) + bias \tag{3.4}$$

On the other hand, floating-point operations (FLOPs) can be divided into two different metrics. First, it can refer to the total arithmetic calculations that are fixed as a hardware feature or capability. Another one refers to the total arithmetic calculation in a neural network model, which will be the one that will be used here. Its calculation is basically the number of parameters without adding the bias multiplied by the image size and a constant of 2 (i.e., representing 1 multiplication and 1 addition), as in (3.5).

$$\text{FLOPs} = 2(C_{out} * C_{in} * K_w * K_h * H_{out} * W_{out}) \tag{3.5}$$

Besides that, there is also a model size, which can be calculated or observed in different ways. The simplest way is to observe the file size of the optimised model in bytes. To do that, it needs to be converted; however, since there are many software tools for model conversion that inhibit model compression ability, this hinders the process of evaluating the optimised model in its purest form. That is why another way is to multiply the number of parameters by a constant 4 (i.e., representing 4 bytes) and divide by 1 MB, as in (3.6).

$$\text{Model Size (MB)} = \frac{4(Num.of\ params.)}{1M} \tag{3.6}$$

Meanwhile, for inference time, it refers to the time it takes for a single image or sample to be classified by the model, as in (3.7). In this case, it excluded any preprocessing, such as model initialisation time, and post-processing time, such as image annotations and labelling. By doing this, the computation cost of different types of hardware (i.e., CPU-based and GPU-based) can be observed and compared in order to justify whether the optimised model still requires further improvements.

$$\text{Inference time (ms)} = current\ time - start\ time \tag{3.7}$$

To conclude, by assessing all five evaluation metrics when deploying the optimised model on the desktop computer hardware, the optimisation strategy can be justified. In theory, based on the formulas above, the number of parameters, FLOPs, model size, and classification more or less depend on the number of parameters, which makes them kind of like model-dependent evaluation metrics. On the other hand, inference time will mostly depend on the clock performance of the desktop computer, whether using a CPU or a GPU. Furthermore, by varying the sparsity level, the relationship between all the evaluation metrics will be clearer in determining the feasibility of the proposed optimisation strategy. Finally, the methodology will continue on the next stage to deploy this and several other optimised models (i.e., using several different image classification tasks) on edge devices for further investigation to determine if the optimisation strategy is able to provide relevant and comparable findings.

3.5 Stage 3: Evaluate the Performance of the Optimised Lightweight CNN Model in terms of Classification Accuracy and Inference Time on Edge Devices

Lastly, in Stage 3 of this research methodology, the improved and optimised model will be evaluated on edge devices. This will help to ensure that the performance of the improved and optimised model will be properly validated as a lightweight CNN model for edge devices by continuing the evaluation of the previous stage on the desktop computer platform. Also, several other image classification tasks will be trained, optimised, and deployed subsequently on the edge devices to further analyse the compatibility of the optimised strategy with the hardware capabilities of the resource-constrained devices. Not only that, but further experimentations will also provide more on the generalisation performance due to the various sizes of the other image classification datasets and applications. After deploying the model on an edge device, the performance of the inference will dictate whether the model is feasible or hyperparameter fine-tuning needs to be configured in order to achieve a comparable performance with the baseline results from the related works. Last but not least, a comparative analysis will be conducted using all the data collected, and various comparisons will be made together with the performance of the recent lightweight CNN model conducted by other researchers.

3.5.1 Deploy the Optimised Model trained with several Image Classification Datasets on Edge Devices

As stated earlier, once the optimisation strategy has been proved on the desktop computer platform, as in Stage 2, the model will be trained, optimised, and deployed for several different image classification tasks on edge devices. Based on that, Figure 3.6 simplifies the required procedures for this subprocess, including the dataset preparation, training settings, and deployment method and configurations.

Primarily, various image classification datasets are identified based on image classification tasks that have been performed by other researchers as part of the process to further validate the optimisation strategy. Intentionally, all the datasets will be varied in terms of their sizes and applications in order to support and measure the generalisation criteria of the optimised strategy towards the optimised model as part of the research objectives. For that, this research has selected a small-sized dataset, a medium-sized dataset, and a large-sized dataset. Additionally, the dataset of image classification for different classes of leaf diseases for a groundnut plant (Manvikar & Reddy, 2023) will be used again on top of some other image classification datasets.

As such, one of them is the type of material waste image classification named TrashNet (Qin et al., 2021), as shown in Table 3.5. This dataset has 2527 of the total sample with 6 classes of waste categories, which include cardboard, glass, metal, paper, plastic, and trash. This is a small dataset on which the application is focused in the waste industry to segregate items for recycling purposes. In depth, this dataset contains 403 images labelled as cardboard, 501 images of glass, 410 images of metal, 594 images of paper, 482 images of plastic, and 137 images of general trash that includes items such as food wrappers, used toothbrushes, rotten fruits, and many others. Also, all images are RGB in the format extension of .jpg with a 512x384 size.

Apart from that, a dataset of the surface of the ball screw drive (Rani et al., 2023) as displayed in Table 3.6 is also included. This dataset contains 21835 images with two types of classes, which are named "P" and "N." “P” stands for pitting, which is basically a surface failure of a ball screw drive, while “N” stands for no pitting, which reflects a good condition for the surface of the ball screw drive in its manufacturing industry that can be utilized for defect detection applications. In addition, this dataset can be considered as a medium-sized dataset with 11075 images for the “N” class and 10760

images for the “P” class. All the images are also taken in RGB with the .png format extension and a size of 150x150.

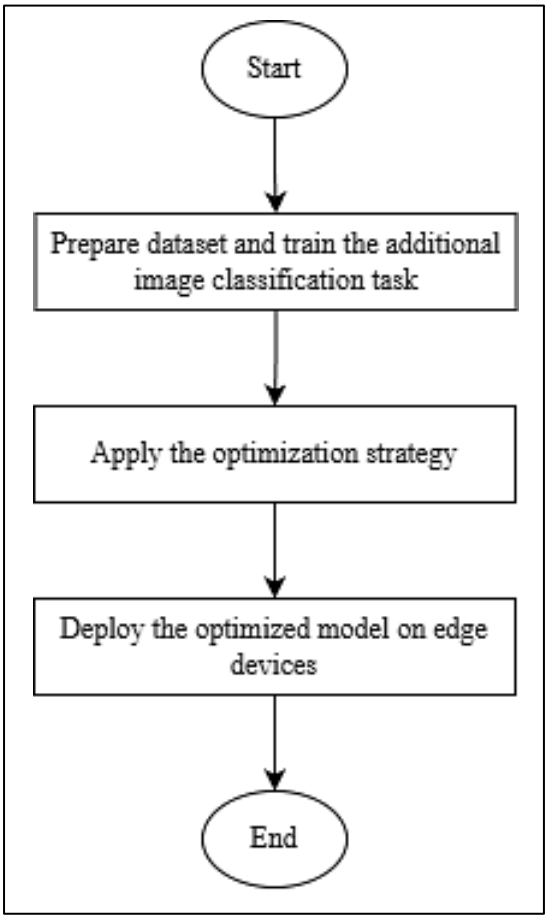

Figure 3.6 Subprocess to Train, Optimise, and Deploy the Additional Image Classification Tasks

Table 3.5
TrashNet Dataset Specifications

Name of the dataset	TrashNet (Qin et al., 2021)
Input size	512x384x3
Number of classes	6
Total images for training and validation	1769
Total images for inference	758

Another multi-class sample dataset, as summarised in Table 3.7, is taken from (Chang & Zheng, 2024), which provides an image classification dataset of concrete surface cracks. This dataset includes 96092 total samples with 2 classes of crack and no-crack images. Originally, this dataset was combined from two datasets, which are “Concrete Crack Images for Classification” (Özgenel, 2019) from Kaggle and

“SDNET2018” (Dorafshan et al., n.d.). Furthermore, the scope of the concrete areas is based on buildings, bridge decks, walls, and pavements in which its application can contribute to safety causes and hazard prevention. As a result, this combined dataset contains 28484 images labelled as "crack" and 67608 images labelled as "no crack." On top of that, all images are also in RGB with the .jpg file extension, yet the input sizes are mixed with 227x227 from (Özgenel, 2019) and 256x256 from (Dorafshan et al., n.d.).

Table 3.6
Ball Screw Drive Dataset Specifications

Name of the dataset	Ball screw drive defect detection (Rani et al., 2023)
Input size	150x150x3
Number of classes	2
Total images for training and validation	15283
Total images for inference	6552

Table 3.7
Concrete Surface Dataset Specifications

Name of the dataset	Concrete surface crack detection (Chang & Zheng, 2024)
Input size	Mixed (i.e., 227x227x3 and 256x256x3)
Number of classes	2
Total images for training and validation	67261
Total images for inference	28831

For the training and validation environment, all additional image classification tasks basically follow the previous setup, as in subsection 3.2.2, which refers to the setup of the Anaconda environment to execute the training and validation based on the first image classification task (i.e., the groundnut leaf disease classification). Moreover, in order to ensure that the model is trained properly, it is important to observe and compare its performance with the model’s performance from the related works. Consequently, the TrashNet dataset is required to be trained with 150 epochs and applied with extensive data augmentations to achieve a comparable accuracy due to its small dataset size. After completing all the training and validation concerning the

different image classification datasets, the model can be optimised and deployed on edge devices.

Following the above dataset preparations and training settings, all the particular models for each image classification task will be optimised before being deployed on edge devices, which are the NVIDIA Jetson Nano and the Raspberry Pi 5. For that, similar environments, including libraries, tools, and inference scripts, will be created according to the testing environment previously on the desktop computer. After that, the optimised model will be deployed and run on both of the edge hardware for each of the image classification datasets.

Regarding the differences between the hardware used, it is technically because of the differences in computational requirements that will be executed on both of those limited-resource hardware. Theoretically, the Jetson Nano that was used earlier in Stage 1 is GPU-based hardware. This means that it features parallel computing rather than traditional sequential computing. On the other hand, the Raspberry Pi 5 is CPU-based hardware that features sequential computing. Thus, by considering that the inference time of any image classification application can be influenced by hardware capabilities such as RAM and the processor of the hardware, both selected hardware will provide more values and comparisons from the impacts of the optimisation strategy.

As some specifications of the Jetson have been summarised in subsection 3.2.3, Table 3.8 lists the features of Raspberry Pi 5 that are related to this experimentation. Basically, unlike the Jetson Nano, which is quite an older version of SoC for AI developments, the Raspberry Pi 5, however, is one of the latest pieces of hardware on the market. This device comes with the Raspbian OS as its OS platform, which is based on the Linux Debian distribution. It employed a Broadcom BCM2712 Quad-Core Arm Cortex-A76 CPU. While it also has a GPU, it is only for lightweight usage, such as video display, and not to be utilised for heavyweight purposes, such as an AI accelerator. Besides that, the Raspberry Pi 5 has many variants with different equipped RAM (i.e., 2 GB, 4 GB, 8 GB, and 16 GB), and the one that will be used for this research is the 16 GB variant. Lastly, the AI environment will be developed using Python and utilised with the PyTorch library and framework.

3.5.2 Assess Comparative Analysis for The Optimised Model Performance Evaluation

After deploying all the optimised models and running inference for each image classification task, the inference performance evaluation results will be collected. Similar to subsection 3.3.2, the inference performance evaluation results will be analysed based on those evaluation metrics. Specifically, due to the additional variables, as in the image classification task variations, the evaluation metrics will be focused more on the classification accuracy and inference time. This consideration is taken into account such that, in one way or another, the sparsity level of the optimisation strategy is related to the number of parameters, FLOPs, and model size. Nevertheless, the classification will be greatly affected by the complexity and capacity of the dataset, while the inference time will be hugely influenced by the computational hardware. With that said, for a more comprehensive assessment of the performance evaluation of the optimised model, apart from the results obtained in this research, the results of related works of each image classification task will be observed and compared as the final assessment. Figure 3.7 illustrates the steps of this subprocess.

Table 3.8
Raspberry Pi 5 Platform Configurations

Device	Raspberry Pi 5
OS Platform	Raspbian OS
CPU	Broadcom BCM2712 Quad-Core Arm Cortex-A76
RAM	16 GB
Environment	Python
Framework	PyTorch

3.6 Summary

In summary, this chapter has constructed the methodology that is necessary to conduct this research. At Stage 1, it defines several sub-processes with the notion of identifying the recent best-performing lightweight CNN model for edge devices in order to achieve the first objective of this study. Next, the several phases of strategies in Stage 2 are planned out to acquire the second objective, which is to optimise the identified lightweight CNN model. Lastly, with the outcome as the third objective, which is to evaluate the optimised model, a few significant strategies are included in Stage 3. Thus,

all the findings will be discussed and justified in the next chapter, which resulted from conducting the research methodology in this chapter.

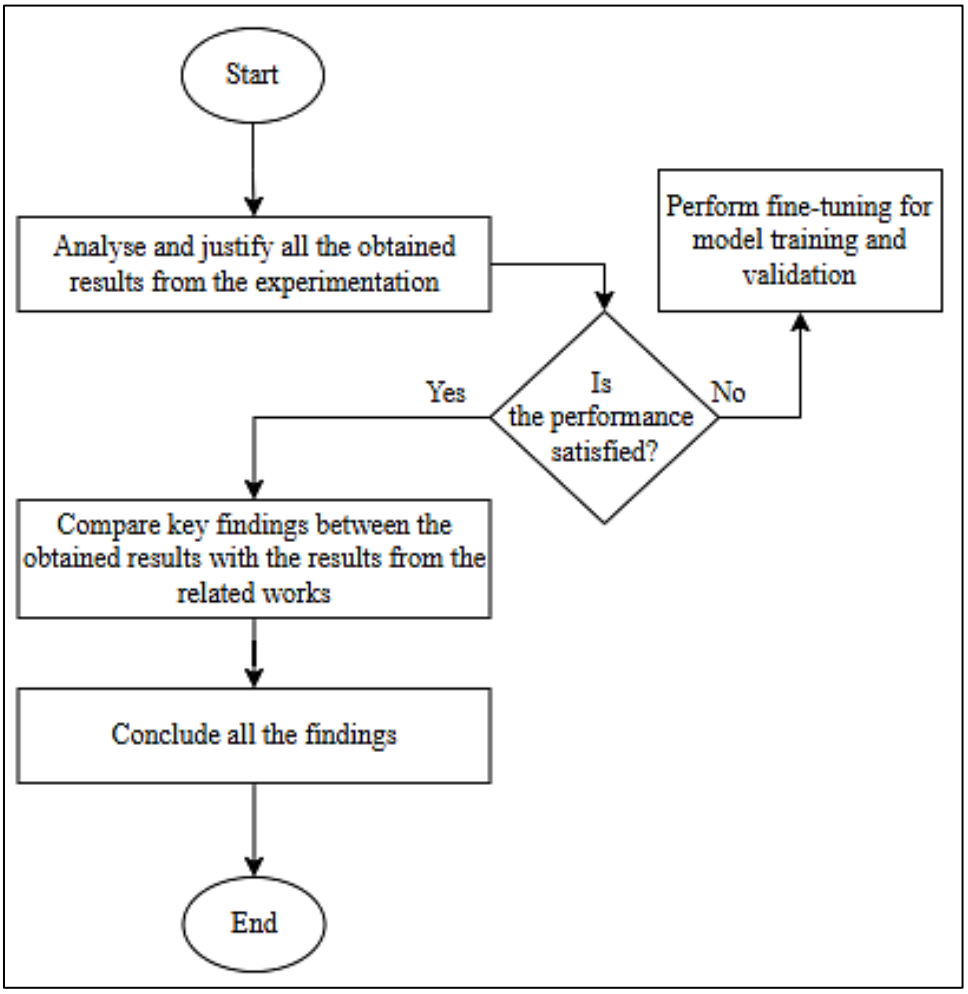

Figure 3.7 Subprocess for Evaluating All the Findings

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Introduction

Based on the methodology outlined in the previous chapter, all the results obtained for each of the respective stages will be presented here. First of all, in order to identify the best-performing current lightweight CNN models, the results of training, validation, and testing or inference for several selected models will be analysed and discussed. Then, after the best lightweight model is identified, the next subsection will observe and explain the inference results based on the optimised model's architecture deployed in a desktop computer environment in order to validate the applied method in optimising the performance of the lightweight CNN model. Last but not least, the inference results of the model executed from the edge devices will be tabulated and justified in the last subsection, as this will help to further clarify the feasibility and performance of the model on edge applications.

4.2 Identification of the Best Performed Current Lightweight CNN Models for Image Classification Task on Edge Devices

As mentioned above, in this subsection, several selected models have been chosen to undergo a strategic method in order to identify the best-performing one of some recent lightweight CNN models. Specifically, several selected models, namely, MobileNetV2, MobileNetV3, ShuffleNetV2, VGG-16, and some of its variants, are first trained and validated using transfer learning and subsequently without transfer learning. On top of that, the dataset used for this test case is based on the groundnut leaf disease dataset. Then, the selected models will be deployed on an edge device (i.e., NVIDIA Jetson Nano), and the resulting performance will be observed and justified. Furthermore, the selected models that were trained using transfer learning will be evaluated first, followed by the models that were trained without using transfer learning.

4.2.1 Training and Validation Result using Transfer Learning for the Selected Models

Initially, for the training and validation results using transfer learning, the performance was evaluated according to four characteristics or metrics, including train loss, validation loss, train accuracy, and validation accuracy, within 35 epochs (i.e., from 1 to 35). With that, this finding can show the efficiency of transfer learning for all of the lightweight convolutional neural network models by observing the consistency of loss reduction and accuracy improvement over epochs. Theoretically, the transfer learning method can provide quick convergence, fast learning, and better generalisation by using pre-trained weights compared to randomly initialised weights. Thus, this method was included to help in identifying the best-performing lightweight CNN model.

By analysing Figure 4.1, which represent the train loss data, it can be seen that VGG-16 obtained the lowest train loss at the final epoch, which was 0.11. This shows that the model has a good feature-learning capacity that can match the training data. Aside from that, ShuffleNetV2 models were shown to have a consistently low and steady training loss across all width multipliers (i.e., from x0.5 to x2.0). This suggests the effectiveness of its channel-splitting and pointwise convolution architecture. For example, larger ShuffleNetV2 variants, such as ShuffleNetV2x1.5 and ShuffleNetV2x2.0, produced results in significantly lower losses (e.g., 0.19 and 0.18) at the last epoch despite having a larger network than their smaller versions that produced comparable losses at 0.23 and 0.19. Hence, this proves the efficiency and stability of its architecture. In comparison, MobileNetV2 converged more gradually to a modest training loss that yielded 0.54 at epoch 35, while MobileNetV3-Small and MobileNetV3-Large converged more slowly and less consistently with fluctuation from early to mid-training and acquired 0.67 and 0.73, respectively, at the end of their training. This further shows that MobileNetV3 needs better hyperparameter tuning or data augmentation to help in adapting its feature learning. Although some models showed a modest or slow training performance in the early stage, all models were able to effectively accelerate their loss during epochs 30 to 31, which indicated their ability to refine specific features for this particular dataset.

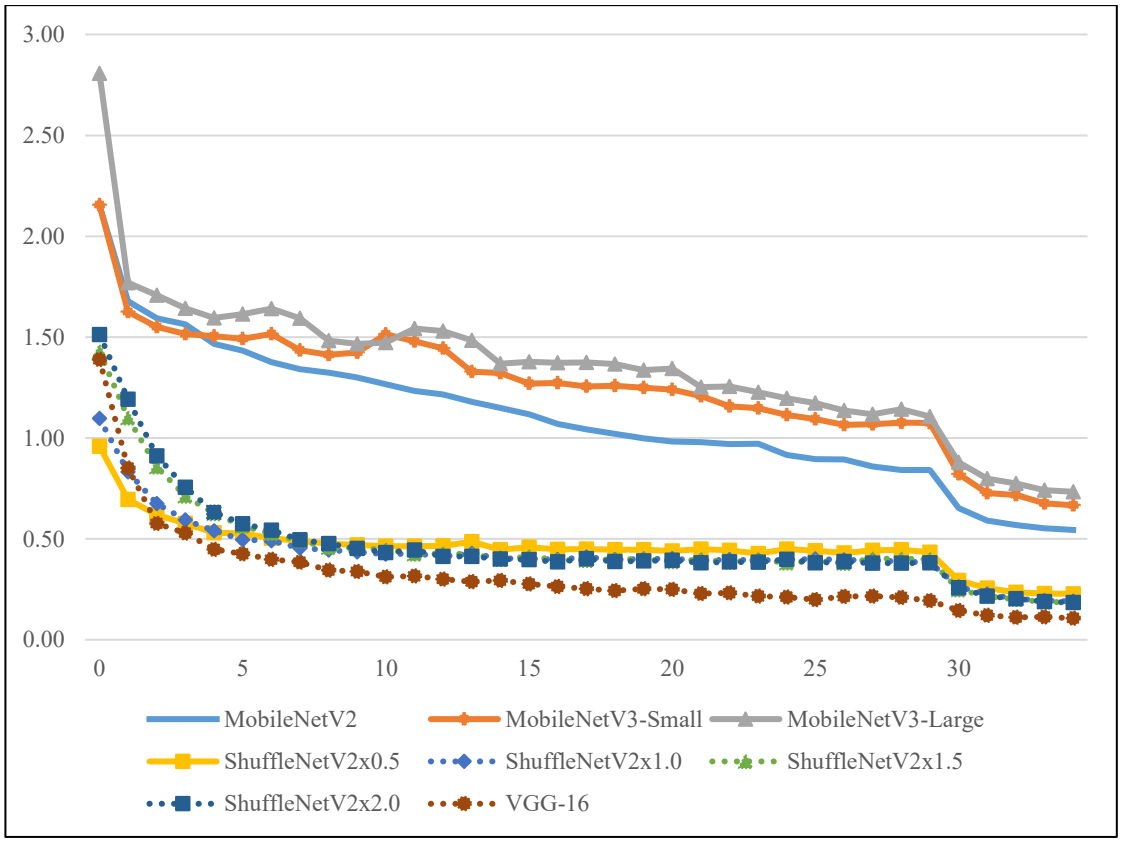

Figure 4.1 Training Loss vs. Epochs using Transfer Learning

Secondly, in terms of validation loss for the training and validation results as shown in Figure 4.2, their trends give further information about each model's generalisability. It can be seen that VGG-16 regularly produced low validation loss that yielded 0.28 at epoch 35, which was expected according to its performance in train loss. Following that, ShuffleNetV2 variations have consistently balanced validation loss during training, especially in later epochs, despite a slight fluctuation before that. This indicates the model's robust generalisation and reduced susceptibility to data fluctuation as the training goes on, which produced final validation loss at 0.51, 0.61, 0.49, and 0.56, respectively. In contrast, MobileNetV3-Small and MobileNetV3-Large showed large variations and occasional sudden spikes in validation loss, which can surpass three times their average values (e.g., MobileNetV3-Small acquired 3.96 validation loss at epoch 10). These variations indicate instability, implying that the squeeze-and-excitation techniques and non-linear activations utilised in MobileNetV3 may be less well suited to the dataset's properties without further tweaking. Nonetheless, despite their earlier performance, both models were able to achieve comparable validation loss to ShuffleNetV2 at the final epoch, which is 0.63 for MobileNetV3-Small and 0.59 for

MobileNetV3-Large. Additionally, MobileNetV2 illustrated a better performance that gave a smoother loss pattern throughout the training and obtained 0.45 validation loss at epoch 35 compared to MobileNetV3.

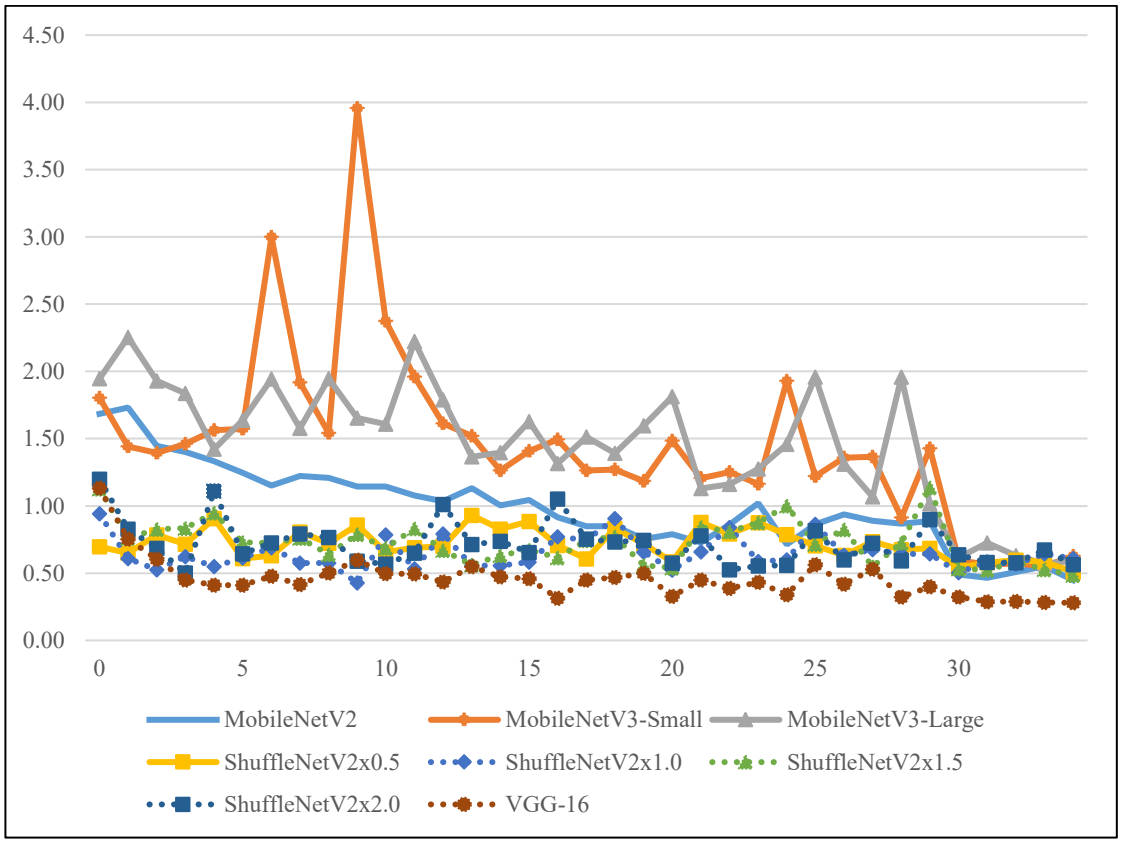

Figure 4.2 Validation Loss vs. Epochs using Transfer Learning

Thirdly, the training accuracy data tabulated in and illustrated in Figure 4.3 show the variations of learning dynamics across all the selected models. As part of it, VGG-16 gets the highest training accuracy, which surpasses 96% by the last epoch, and this behaviour suggests that its outstanding fitting capabilities, as expected, are observed from its training loss and validation loss. On top of that, ShuffleNetV2 models also showed an excellent training accuracy very early on that exceeded 80% during the first 10 epochs and finally achieved 92.08%, 93.20%, 93.48%, and 93.67% accuracy with respect to each of the model's variations. This further demonstrates the efficient gradient flow and good usage of pre-trained features of the model. Other than that, MobileNetV2 improved steadily but eventually reached an average final training accuracy of 80.26%, while MobileNetV3-Small and MobileNetV3-Large were a bit unstable and lagged, attaining the lowest training accuracy even after lengthy training

compared to the others. For example, MobileNetV3-Small and MobileNetV3-Large are only able to yield their accuracy at 74.94% and 72.84% at the last epoch, respectively. As such, this pattern indicates that MobileNetV3 architectures may need more aggressive fine-tuning procedures or larger datasets in order to fully leverage their design advantages.

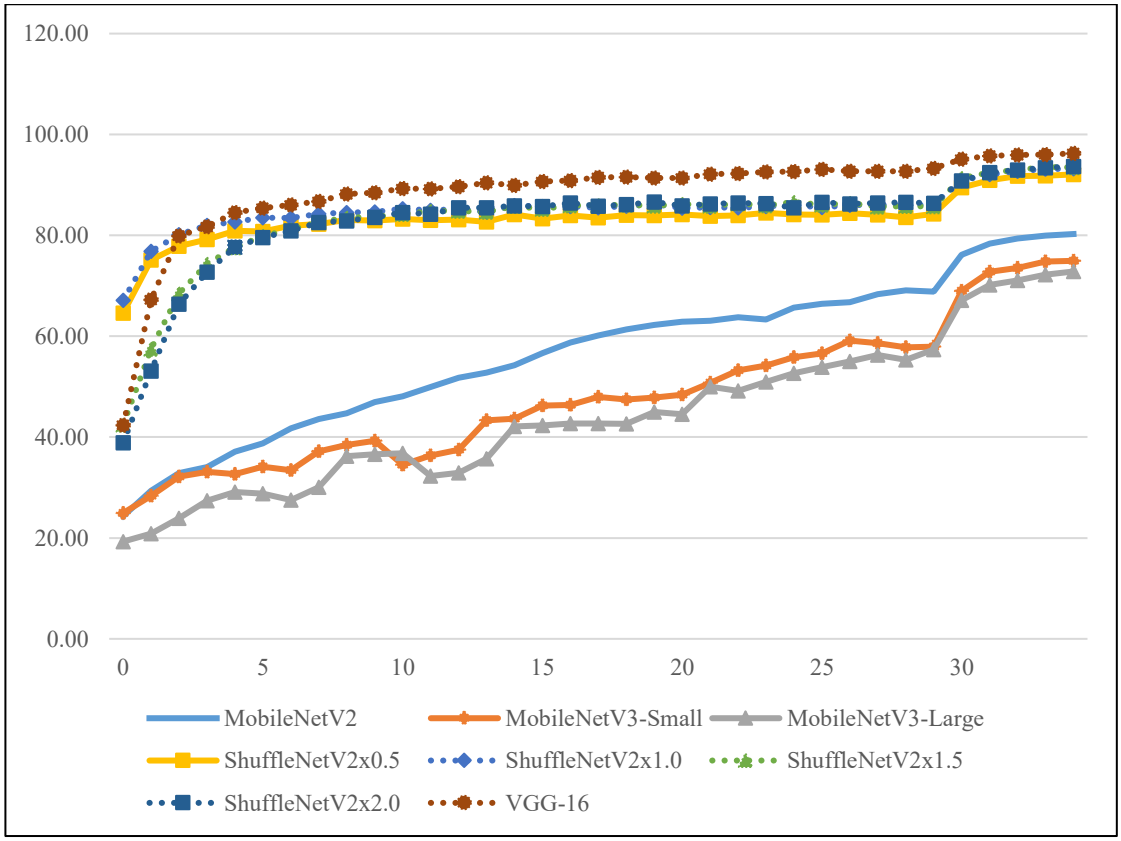

Figure 4.3 Training Accuracy (%) vs. Epochs using Transfer Learning

Last but not least, validation accuracy is also one of the most important criteria for assessing real-world performance of the selected models, where the findings can clearly distinguish their behaviour for selection as shown in and Figure 4.4. For the most part, VGG-16 had the greatest final validation accuracy obtained at 89.37%. This further proves that the model displays excellent classification capabilities and proves its efficacy when accuracy is the major goal. Nevertheless, ShuffleNetV2 models attained validation accuracies comparable to VGG-16 while having much reduced computational complexity. For instance, the variations of the model acquired 84.53%, 85.44%, 85.77%, and 85.86% of validation accuracy at epoch 35 for x0.5 to x2.0, respectively. This demonstrates ShuffleNetV2's superior balancing accuracy and

efficiency, even though its variations have a different number of parameters. Meanwhile, MobileNetV2 shows a pattern rather closer to linear and has reasonable validation accuracy, which yielded 83.16% at the last epoch. On the contrary, MobileNetV3-Small and MobileNetV3-Large routinely underperform other designs due to instability in validation loss and worse performance from the early stage to around epoch 30. As a result, the procured final validation accuracy was at 78.77% and 76.14% with respect to both variations.

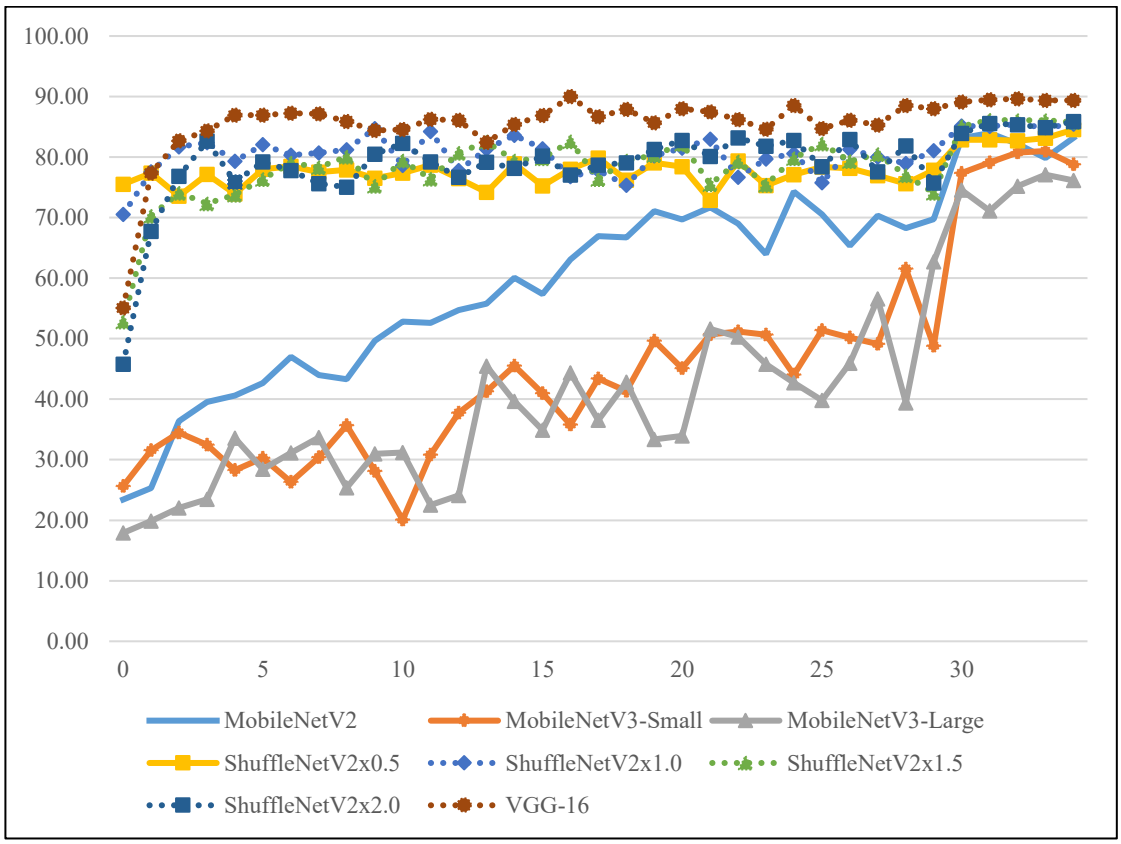

Figure 4.4 Validation Accuracy (%) vs. Epochs using Transfer Learning

Based on the observed disparity between training and validation performance using transfer learning, it gives valuable insight into each selected model's behaviour. For one, VGG-16 has demonstrated a tendency to slightly overfit. This is because, despite its excellent training accuracy and lowest loss for both training and validation, its validation accuracy is still comparable to ShuffleNetV2. ShuffleNetV2 models, on the other hand, have a smaller gap, indicating higher generalisation and more efficient model capacity utilisation. In comparison, while MobileNetV2 shows steady and moderate performance, MobileNetV3 models have irregular gaps due to unstable

validation behaviour, which increases their susceptibility to training (i.e., hyperparameters and data augmentations) configurations. Overall, ShuffleNetV2 models are interpreted as having the most respectable and well-balanced performance across their model's variations.

4.2.2 Training and Validation Result without using Transfer Learning for the Selected Models

As for the training and validation findings obtained without transfer learning, it shows significantly different learning dynamics than in the transfer learning context. Typically, when trained from scratch, all types of CNN models should show slower convergence and learning rates, as well as greater initial loss values, as opposed to the transfer learning method. This behaviour reflects the greater difficulty of learning from limited or task-specific data as compared to fine-tuning with pre-trained weights. An observation of the reduction in loss and improvement in accuracy trends will further reveal the performance of all models in terms of their capability to extract features such as textures and edges from the training images. Simply said, this will demonstrate how well they can acquire discriminative features even without transfer learning, as this will help to increase their feasibility and robustness that can be deployed to suit any type of edge application for image classification.

Among the architectures, all designs exhibit a slow but steady convergence in the early epochs and greater initial loss values, as depicted in Figure 4.5. In addition, there was an obvious significant loss around epoch 30 to 31. It can be seen that the bigger the width multiplication of ShuffleNetV2 models, the lower the training losses that their variants can yield. For example, from x0.5 to 2.0, the variants obtained 0.29, 0.25, 0.24, and 0.23 of train loss at epoch 35. This means that the increasing width enhances representational capacity without compromising training stability. VGG-16, on the other hand, retains a relatively greater training loss for most of the training phase and is only able to achieve 0.27 training loss at the final epoch following ShuffleNetV2. This further demonstrates the difficulties of optimising a deep, parameter-heavy architecture while being trained from a randomly initialised weight. Meanwhile, MobileNetV2 and MobileNetV3 illustrated a rather moderate result, which also slowly converged to produce train losses of 0.41, 0.32, and 0.35, respectively, at the last epoch. Overall, ShuffleNetV2 variants outperform MobileNetV2, MobileNetV3, and VGG-16

in terms of loss reduction and final training loss and thus demonstrate that their lightweight design and effective feature reuse make optimisation possible even when initialised randomly.

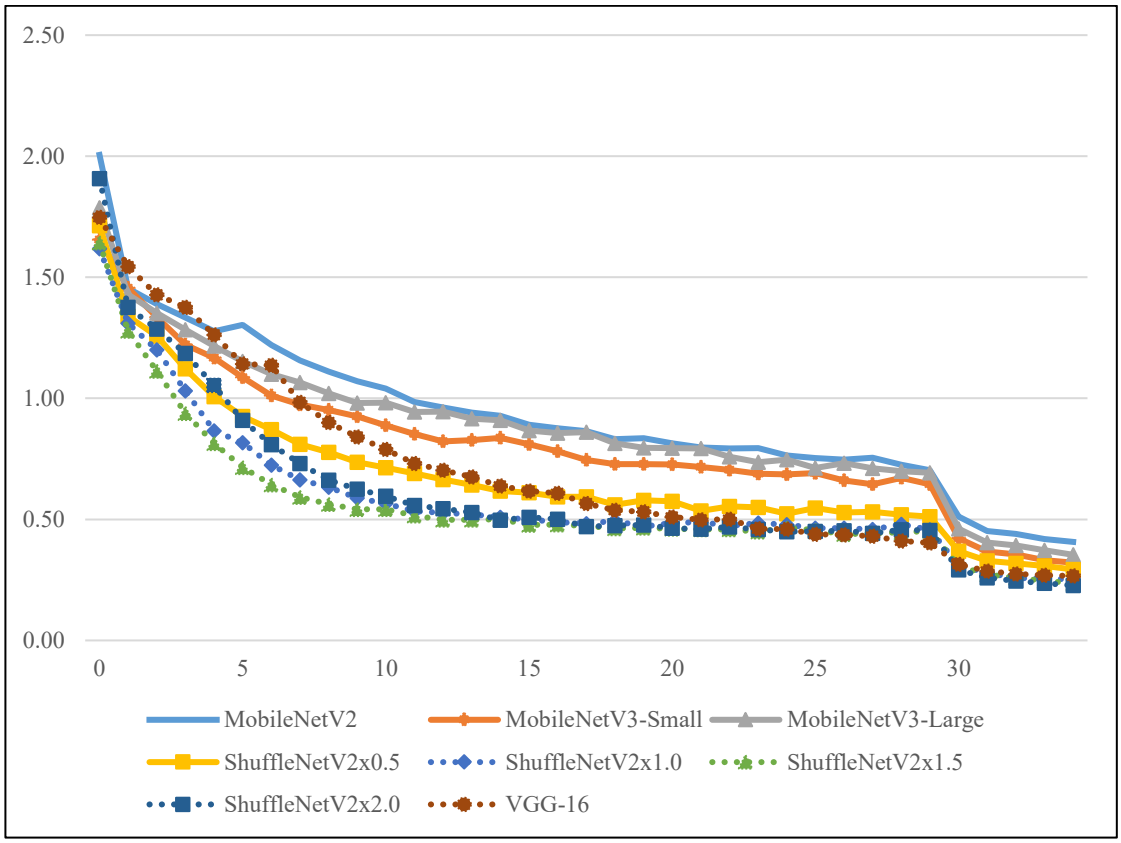

Figure 4.5 Training Loss vs. Epochs without using Transfer Learning

Moreover, the validation loss patterns highlighted in Figure 4.6 exhibit the difficulty of training without transfer learning. The validation loss varies significantly, differing between designs of the selected models, with occasional dramatic spikes, especially in the middle epochs. For the most part, MobileNetV2 was able to achieve the lowest loss at 0.41 for the final epoch, and concurrently, it was the same result for its train loss. This implies that its model had a good fit for training and validation while being trained from scratch. Other than that, VGG-16's validation loss decline was rather smooth in later epochs, although it remains larger than that of other lightweight models. This indicates that the model had poor generalisation efficiency relative to model complexity and capacity, which only yielded 0.47 validation loss at epoch 35. Meanwhile, MobileNetV3 shows a noticeable instability where sudden spikes in validation loss further describe its poor sensitivity to weight initialisation and probable

overfitting to the training data. As such, both variants only managed to obtain 0.48 and 0.52 validation loss at the final epoch. Similarly, in spite of its models having a steady validation loss trajectory through its training, ShuffleNetV2, particularly the x0.5 variant, also exhibits a tremendous fluctuation of 3.21 at epoch 9. Aside from that, the variants are able to produce loss at 0.58, 0.55, 0.49, and 0.61 for epoch 35.

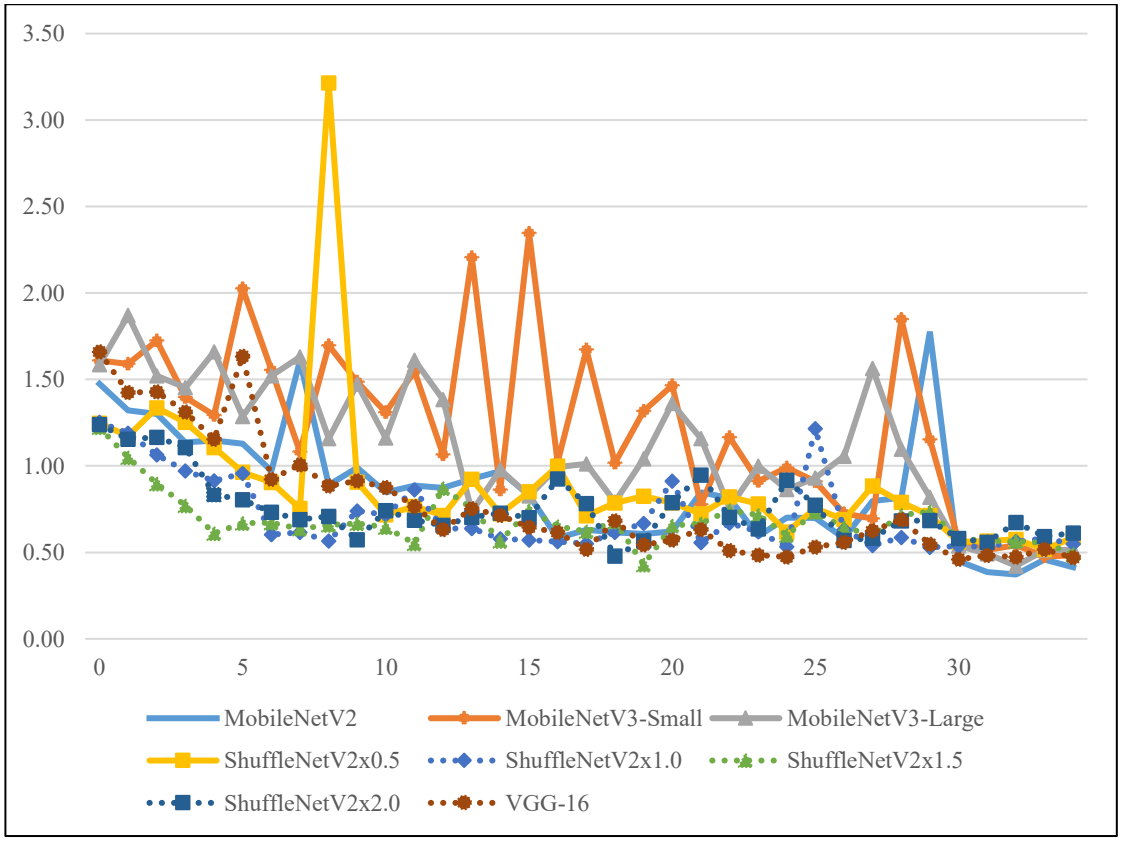

Figure 4.6 Validation Loss vs. Epochs without using Transfer Learning

Next, training accuracy trends displayed in Figure 4.7 show that all models gradually increase in classification performance in a smooth and steady state. The ShuffleNetV2 model obtained the greatest final training accuracy among the other lightweight networks, at 89.47%, 91.06%, 91.56%, and 91.92% of training accuracy for all of its variants. This indicates the model has significant learning capability even with random initialisation. Following that, VGG-16 exhibits a delayed and slower but continuous increase in training accuracy and eventually reaches competitive performance with 90.85% training accuracy. On the other hand, MobileNetV2 and MobileNetV3 designs show continuous but modest advances, with MobileNetV3 eventually outperforming MobileNetV2 with 88.48% and 87.46% of train accuracy

from MobileNetV3 compared to 85.28% from MobileNetV2 at the final epoch. This is because MobileNetV3 has a larger representational capacity for feature extraction from the training images. For the most part, ShuffleNetV2 variations generally beat MobileNetV2, MobileNetV3, and VGG-16 designs in the early and mid-training stages, resulting in greater training accuracy with fewer epochs.

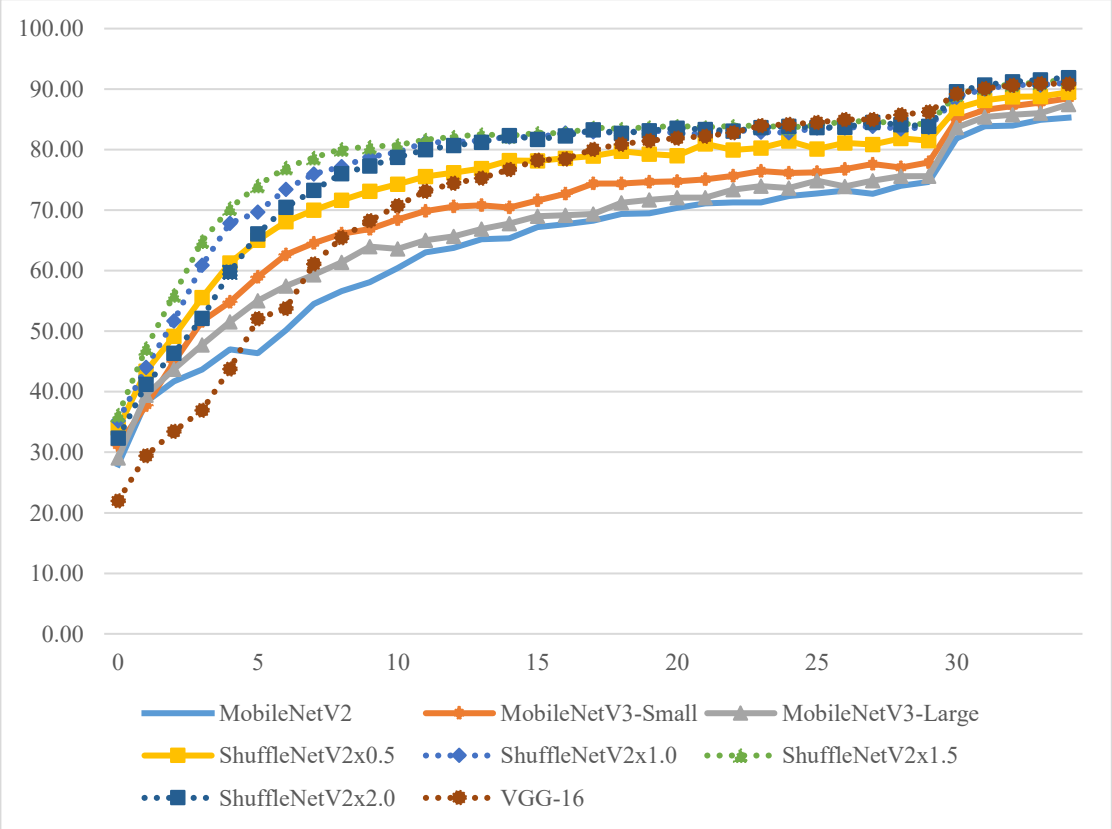

Figure 4.7 Training Accuracy (%) vs. Epochs without using Transfer Learning

Additionally, as the selected models' validation accuracy is highlighted in Figure 4.8, it shows an inconsistent rise and stays much lower throughout the early epochs for all models, especially for MobileNetV2, MobileNetV3, and VGG-16. Ultimately, most of the models obtained comparable final validation accuracy at the last epoch. For ShuffleNetV2, it can be said that the variants have acquired a moderately consistent validation accuracy at 83.23%, 85.42%, 85.82%, and 86.15%. This implies its resilience and appropriateness by being trained from randomly initialised weights. Aside from that, although VGG-16 eventually achieves competitive validation accuracy for epoch 35 at 86.34%, it takes many more epochs to reach around the same high accuracy. Similarly, MobileNetV2, which yielded 87.50% of validation accuracy, also

shows adequate generalisation in later epochs. Meanwhile, both MobileNetV3-Small and MobileNetV3-Large were able to display their validation accuracy at 84.03% and 83.73% at the final epoch, despite having more fluctuations throughout this phase, which supports their poor stability compared to the others.

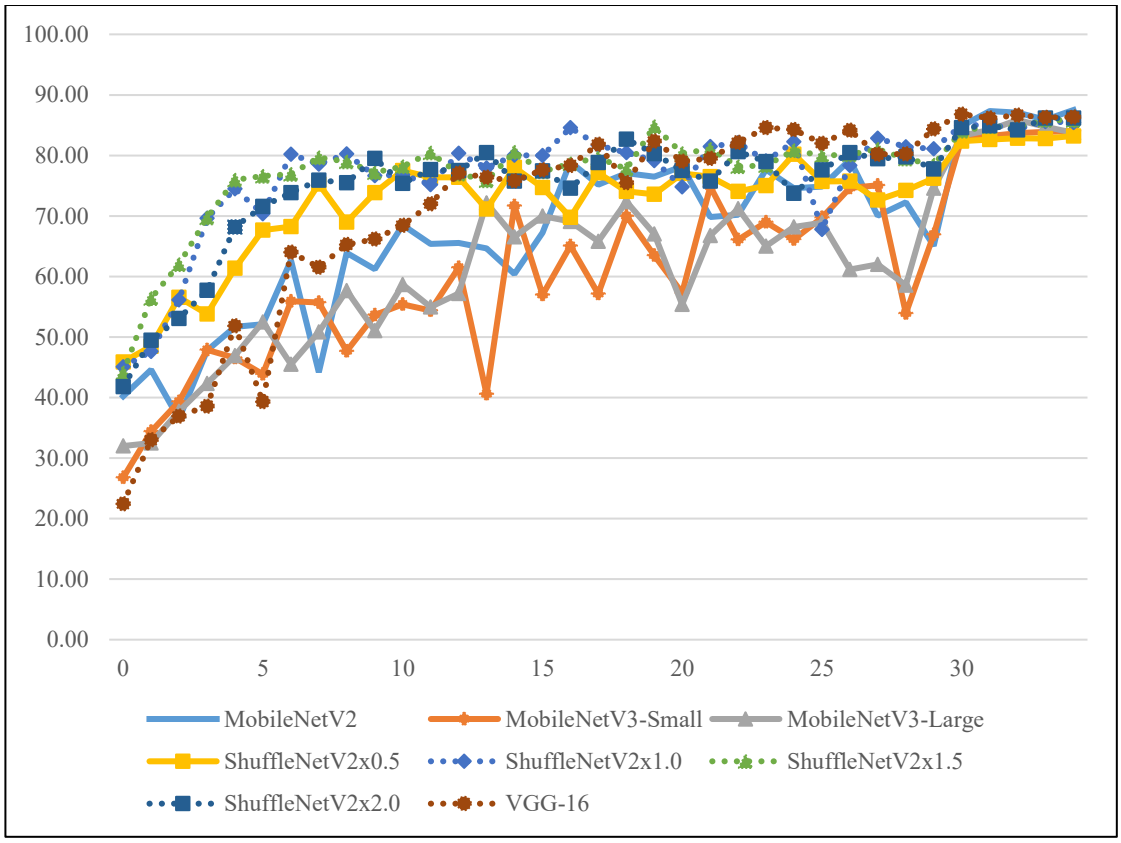

Figure 4.8 Validation Accuracy (%) vs. Epochs without using Transfer Learning

Overall, the results show that training convolutional neural networks without transfer learning greatly increases optimisation difficulty and decreases generalisation efficiency for some of the selected models. Moreover, it can be seen that there is a contrast of patterns between the training and the validation phase. Although the training phase had steadier and smoother trends, the validation phase reflected otherwise. This is because the limited dataset and the absence of pre-trained weights further pushed the difficulty of the classification process during feature mapping. While most of the models reacted in the same way, some models also illustrated some effort to adapt to the training and validation without transfer learning. For example, MobileNetV2 and MobileNetV3 variations show many obvious spikes in validation accuracy as opposed to their training accuracy. However, ShuffleNetV2 architectures had fewer spikes in

validation accuracy, which implies greater regularisation and more efficient model capacity use despite random initialisation. Therefore, while this effect can be mitigated by applying more data augmentation, reducing the learning rate and many others, this behaviour also demonstrates the inherent effectiveness of the ShuffleNetV2 architecture in facilitating generalisable feature learning even when this particular model was being trained with random weight initialisation.

4.2.3 Inference Result of the Transfer Learning Models

In regard to the selected models that have been trained using transfer learning, the inference results achieved by deploying the models on the NVIDIA Jetson Nano shed light on the trade-off between the performance of the selected models in a resource-constrained environment. This is because the Jetson Nano has limited GPU resources and memory bandwidth, thus inference speed is as critical as classification accuracy. The results clearly show that architectural efficiency is the most important factor in determining real-time performance, even when models have equivalent classification accuracy during training and validation assessment. Table 4.1 depicts the inference results obtained for classification accuracy and inference time on the edge device (i.e., Jetson Nano), which has varied performance for all of the selected models.

Table 4.1
Jetson Nano Classification Accuracy and Inference Time using Transfer Learning

Model	Accuracy (%)	inference time (ms)
MobileNetV2	91.02	14.50
MobileNetV3-Small	87.40	10.63
MobileNetV3-Large	82.88	14.15
ShuffleNetV2x0.5	**98.06**	**9.41**
ShuffleNetV2x1.0	97.66	12.30
ShuffleNetV2x1.5	97.21	14.80
ShuffleNetV2x2.0	95.62	17.12
VGG-16	95.84	**82.40**

Among all tested architectures, the ShuffleNetV2 model consistently provides the greatest balance of accuracy and inference speed. ShuffleNetV2x0.5 obtains the maximum classification accuracy of 98.06% while simultaneously having the shortest inference time of 9.41 ms, which displayed an outstanding efficiency on the Jetson

Nano. This result further justifies the effectiveness of ShuffleNetV2's channel-splitting and pointwise convolution method, which reduces memory access costs and processing overhead. As the width multiplier grew from x0.5 to x2.0, their inference time steadily increased from 9.41 ms, 12.30 ms, 14.80 ms, and 17.12 ms, respectively, due to the increase in the number of parameters. Nevertheless, while the yielded inference time came as expected, their classification accuracy showed otherwise, with decreased accuracy from 98.06%, 97.66%, 97.21%, and 95.62% with respect to their variants. This trend implies that with the rising model width, the transfer learning method finds it more difficult to fit the training data with feature mapping with the current hyperparameters or data augmentation configurations.

Aside from that, MobileNetV2, with a validation accuracy of 91.02% and an inference time of 14.50 ms, indicated a good efficiency while having much lower accuracy than ShuffleNetV2 models. Yet MobileNetV3-Small, which has the quickest inference time of the MobileNetV3 variants at 10.63 ms, yielded a lower accuracy of 87.40%. Additionally, MobileNetV3-Large also achieved just 82.88% accuracy with an inference time of 14.15 ms, which implies neither greater accuracy nor faster inference speed compared to the other MobileNet models. As a result, this finding shows that MobileNetV3 has greater architectural complexity, which includes squeeze-and-excitation blocks and non-linear activation, and may not be suitable and efficient for this particular application and may need another method to fully utilise its architecture.

In contrast to the other lightweight CNN models, VGG-16 had a different performance profile in terms of its inference time despite having a competitive classification accuracy. Specifically, although it had a comparatively high accuracy of 95.84%, its inference time was substantially longer at 82.40 ms, which renders its unsuitability for a real-time application on embedded devices where inference time is critical. The high delay was due to VGG-16's deep architecture and huge number of parameters, which caused excessive memory transfers and computational burden. This finding highlights the limitations of classic high-capacity convolutional neural networks when implemented on resource-constrained edge devices, notwithstanding their great classification performance.

To conclude, this analysis of accuracy and inference time highlights some of the advantages of the selected lightweight models using transfer learning in an edge device environment. For the most part, ShuffleNetV2 stands out as the best model for real-time inference, providing greater accuracy, low latency, and optimal resource consumption.

In contrast, VGG-16 represents the least efficient extreme, with inference time significantly longer than that of the other models. Nevertheless, MobileNet models, including MobileNetV2 and MobileNetV3, fall somewhere in the middle, but they cannot match ShuffleNetV2's accuracy and latency performance. Overall, the inference findings based on the transfer learning method show that the use of lightweight, hardware-efficient architectures such as ShuffleNetV2 was more relevant when selecting a model for edge deployment.

4.2.4 Inference Result of the Models without Transfer Learning

In conjunction with the inference results from transfer learning, the inference results obtained from deploying the selected models that were trained with randomly initialised weights on Jetson Nano further provided useful information on how the training strategy affects the deployment performance. Unlike with transfer learning, where pre-trained representations dominate performance, models learned from scratch rely only on data-specific learning for feature extraction. Although network design and hardware parameters heavily influence inference time, classification accuracy, however, reflects the efficacy of learnt representations and their appropriateness for real-time deployment on resource-constrained devices. With that, Table 4.2 summarises the inference results based on classification accuracy and inference time.

Ultimately, when trained without transfer learning, it can be seen that the ShuffleNetV2 model performed well and consistently in inference. Expectedly, ShuffleNetV2x2.0 had the best classification accuracy of 98.64% but the highest latency among the ShuffleNetV2 variants at 17.22 ms due to its greater network compared to the other variants. Specifically, while this inference time was still within realistic bounds for many real-time applications, the increase in latency as compared to smaller variations reflects the projected cost of increased model size. Additionally, ShuffleNetV2x0.5 and ShuffleNetV2x1.0 strike a good mix between accuracy and inference speed with accuracies around 96% and inference durations around 14-15 ms. Similarly, while not showing much improvement, ShuffleNetV2x1.5 also obtained a comparable result of 96.75% accuracy with 16.72 ms inference time. For that, these results show that ShuffleNetV2 architectures can learn strong feature representations from scratch while keeping efficient inference characteristics on edge applications.

In terms of accuracy, both MobileNetV2 and MobileNetV3 models obtained classification accuracies of 95.70%, 95.30%, and 95.99%. This indicates that training from randomly initialised weights enables these models to better adapt to the dataset's individual properties, such as the edges and textures of the training images. Moreover, in terms of inference time, MobileNetV2 and MobileNetV3-Small achieved 14.56 ms and 14.73 ms, which were comparable to ShuffleNetV2x0.5. Significantly, MobileNetV3-Large also yielded 15.54 ms, which was comparable to ShuffleNetV2x1.0. Nevertheless, while these MobileNetV2 and MobileNetV3 models cannot outperform ShuffleNetV2 variations in terms of speed or accuracy, their complicated architectural components displayed a modest efficiency improvement in this deployment configuration.

In comparison, VGG-16 had the slowest inference time at 78.98 ms and the lowest classification accuracy of 93.51%. This implies that the poor suitability of the model for real-time inference on the Jetson Nano and, in addition, its outperformance in terms of classification accuracy further support that criterion. Furthermore, the high latency for VGG-16 demonstrates the influence of large convolutional kernels and deep feature maps on memory access and computation overhead in embedded GPU hardware on Jetson Nano. Meanwhile, its low classification accuracy despite having high training and validation accuracy also indicates that the model was overfit during the training phase based on the hyperparameters and data augmentation configuration.

In short, this comparison of lightweight models that are being trained without transfer learning can further justify the performance of those architectures in terms of edge applications. This impact was particularly visible in ShuffleNetV2, which continues to offer the best balance of accuracy and speed, with bigger variations benefiting from better accuracy at the expense of increasing latency. Meanwhile, the MobileNet variations provide more predictable and consistent inference performance with just minor decreases in accuracy. VGG-16, on the other hand, was unsuitable for embedded deployment due to its long inference time. All in all, the inference findings show that a lightweight model such as ShuffleNetV2 can achieve comparable classification accuracy and inference time even when its weights are being trained from random initialisation, which exhibited its robustness and suitability for resource-constrained devices.

Table 4.2
Jetson Nano Classification Accuracy and Inference Time without using Transfer Learning

Model	Accuracy (%)	inference time (ms)
MobileNetV2	95.70	14.56
MobileNetV3-Small	95.30	14.73
MobileNetV3-Large	95.99	15.54
ShuffleNetV2x0.5	96.83	14.23
ShuffleNetV2x1.0	96.75	15.34
ShuffleNetV2x1.5	96.37	16.72
ShuffleNetV2x2.0	**98.64**	17.22
VGG-16	93.51	**78.98**

4.2.5 Performance comparison for the Current Lightweight CNN Models

This subsection provides a comprehensive evaluation of multiple lightweight convolutional neural network architectures under both transfer learning and non-transfer learning methods. Besides that, most of the performance was assessed in terms of training loss, validation loss, training accuracy, and validation accuracy under the pretext of the training phase, as well as classification accuracy and inference time in the context of deployment on the NVIDIA Jetson Nano. By examining these evaluation metrics together, the findings give a comprehensive understanding of how architectural design and training methods impact practical performance in resource-constrained edge applications. Hence, by observing and analysing the experimentation on this stage, the best recent lightweight CNN model will be identified for the image classification task on edge devices.

In a model-to-model comparison, ShuffleNetV2 was the most consistently well-performing architecture across all assessment criteria. During the training phase, ShuffleNetV2 can always be seen to outperform other models, such as MobileNetV2 and MobileNetV3, while keeping a competitive result with VGG-16, even when the ShuffleNetV2 models were being trained with randomly initialised weights. In terms of classification accuracy and inference time for testing on Jetson Nano, ShuffleNetV2 also provided a satisfactory performance while maintaining a small gap between the results based on the transfer learning and without the transfer learning method and thus making it ideal for real-time embedded applications. For instance, overall,

ShuffleNetV2 successfully achieved a balance between its classification accuracy, with more than 95%, and less than 18 ms of inference time for both training methods.

On the other hand, MobileNetV2 and especially MobileNetV3 displayed a bigger difference between the two methods of training, resulting in these models seeming to perform better when they were being trained without transfer learning. This was reflected in the results of less fluctuation and better generalisation during their training phase, and the high gain of classification accuracy during the testing phase. Unlike the MobileNet models, VGG-16, however, responded otherwise. While the results delivered were still comparable, it can be seen that VGG-16 performed better with transfer learning based on the classification accuracy itself. However, its deployment performance on the Jetson Nano was severely constrained by excessive inference time, which was more than four times higher than that of the other lightweight architectures, which are clearly not suited for time-critical applications.

In short, across all assessed parameters with both training methods, ShuffleNetV2 was the most effective and adaptable design in providing improved accuracy-efficiency trade-offs, the smallest generalisation gaps, and consistent as well as reliable performance on embedded hardware. On top of that, even without pretrained weights, lightweight architectures like ShuffleNetV2 showed sensible performance, unlike MobileNet models that seem to favour more on non-transfer learning, while VGG-16 seems to incline more towards transfer learning. With that, by highlighting the necessity of considering training method, architectural design, and deployment restrictions when choosing deep learning models for real-world edge computing applications, out of all the selected models, ShuffleNetV2 is the most balanced and practical model for embedded deep learning on the Jetson Nano.

4.3 Optimised the Performance of the Identified Lightweight CNN Models using Appropriate Optimisation Strategies

The previously obtained result on identifying the current best-performing lightweight CNN model had proven that the ShuffleNetV2 was more advantageous compared to the other models. For that, this subsection will provide some results on the improvement that has been made to the ShuffleNetV2. Based on the methodology specified, the ShuffleNetV2, which includes its four variants, notably ShuffleNetV2x0.5, ShuffleNetV2x1.0, ShuffleNetV2x1.5, and ShuffleNetV2x2.0, was

optimised by reducing its number of channels. In addition, this technique was executed first on a desktop computer platform in order to evaluate its feasibility before moving to deploy the models on edge devices, as this will further help to clarify the benefits of this optimisation strategy. Therefore, the following results will be analysed and discussed based on the inference performance on both CPU-runtime and GPU-runtime of the desktop computer environment configurations.

4.3.1 Inference Performance Evaluation of the Optimised Models on a Desktop Computer

Ultimately, the inference performance research conducted on the desktop computer platform using the groundnut leaf disease dataset provided a deep insight into how an organised channel removal influenced model efficiency in terms of the number of parameters, floating-point operations, model size, classification accuracy, and inference time across multiple ShuffleNetV2 variations. The percentage of channel removal, also known as the sparsity level, was the primary variable in the optimisation strategies. By gradually removing the number of channels or filters from the neural network, the relationship among model compactness, computational complexity, and real-world inference performance on both CPU and GPU can be investigated. Hence, this will provide several useful findings for the applied optimisation strategies on the models' behaviour.

Based on Table 4.3, it can be seen that a moderate channel removal (i.e., between 10% and 25%) on ShuffleNetV2x0.5 yielded a highly favourable balance between accuracy and efficiency. In terms of the number of parameters, FLOPs, and model size, the observed performance showed a great advantage, as it succeeded in reducing all those evaluation metrics. For example, the number of parameters can be reduced from 347K to 44K, the FLOPs were also reduced from 43.55M to 2.78M, and the model size managed to be decreased from 1.36MB to 0.17MB. Moreover, with a small reduction in the number of parameters, FLOPs, and model size at around 25% of channel removal, the model can maintain and slightly increase its classification accuracy from 98.49% to 98.65%. This unexpected improvement implies that a mild channel trimming can act as an implicit normaliser that reduces unnecessary channels and improves feature discrimination.

Table 4.3
Inference Results of ShuffleNetV2x0.5 Characteristics and Performances on Desktop Computer

Sparsity Level (%)	Number of Parameters	FLOPs	Model Size (MB)	Accuracy (%)	Inference Time, CPU (ms)	Inference Time, GPU (ms)
0	348K	43.55M	1.36	98.49	15.5	13.91
10	310K	10.50M	1.19	98.49	15.27	15.06
20	277K	9.53M	1.06	98.57	14.66	15
25	261K	9.05M	1.00	98.65	14.55	15.05
30	**244K**	**8.57M**	**0.94**	**96.74**	**14.4**	**15.02**
40	211K	7.60M	0.81	83.11	14.36	15.06
50	177K	6.64M	0.68	58.22	13.95	15.07
60	143K	5.68M	0.55	16.73	13.4	14.64
70	110K	4.71M	0.42	16.69	13.42	14.52
80	77K	3.75M	0.30	16.69	13.08	14.65
90	44K	2.78M	0.17	16.69	**12.17**	**14.99**

However, beyond 30% channel removal, accuracy rapidly declined (e.g., accuracy decreased from 96.74% at 30% sparsity and fell below 60% at the 50% sparsity level). Additionally, as illustrated in Figure 4.9, CPU inference time progressively decreased from 15.27 ms to 12.17 ms as the percentage of removed filters increased, while GPU inference time remained relatively constant across all levels, ranging from a minimum of 15.06 ms to 14.99 ms. Even more, the original model (i.e., with 0% sparsity level) that was executed on the GPU had the lowest inference time of 13.91 ms. This effect reflects that the CPU behaved as expected with lower computational strain in comparison to the GPU execution, which was less susceptible to these optimisation strategies.

Secondly, the ShuffleNetV2x1.0 model followed a similar trend but was more sensitive to the sparsity level because of its higher number of parameters. Table 4.4 presents the model's original capacity, including its number of parameters, around 1.26M, 151.69M FLOPs, and model size, with 4.87MB. Consequently, with continuous channel removal, it shows that the accuracy can stay generally steady up to around 30% of channel elimination (i.e., with 882K number of parameters) with a small decrease in classification accuracy from 96.74% to 95.19%. With that, the FLOPs also decreased to 20.47M, and the model size reduced to 3.38MB. However, after channel removal approaches 40%, the classification accuracy rapidly degrades and drops below 90%.

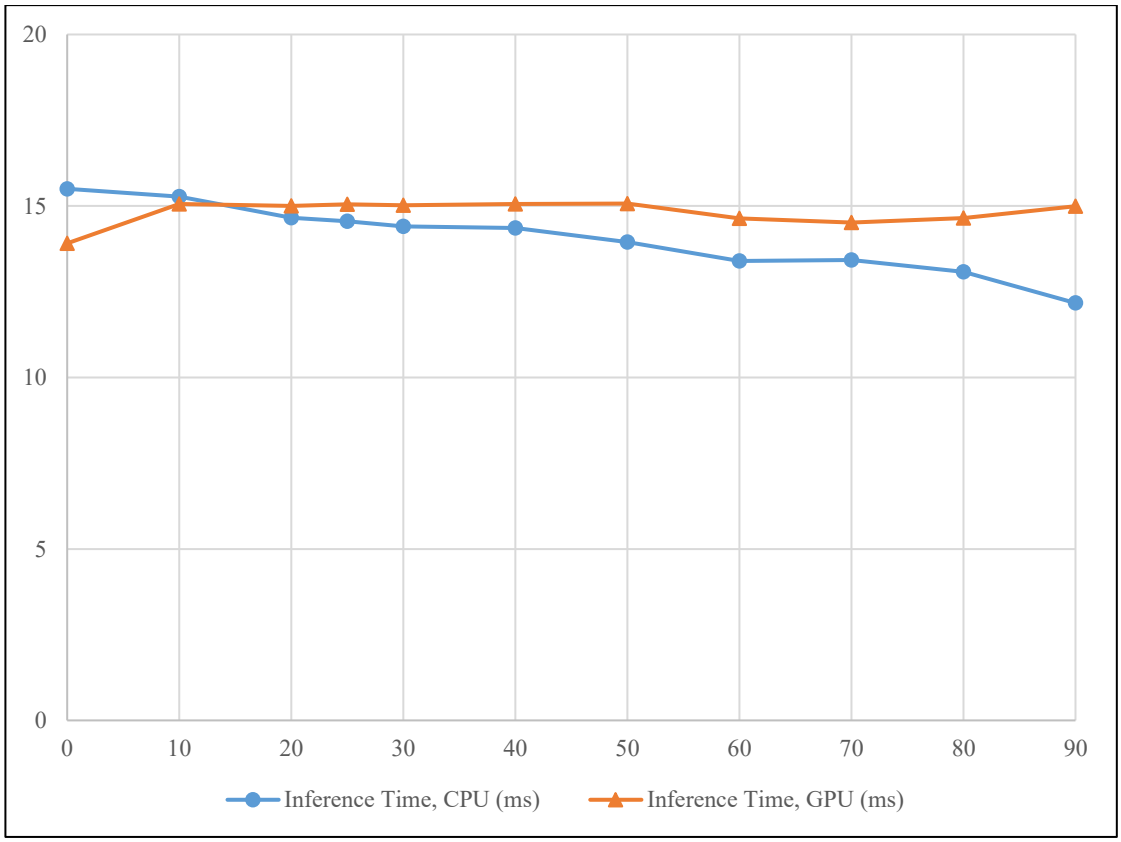

Figure 4.9 Sparsity Level (%) vs. Inference Time on CPU and GPU of ShuffleNetV2x0.5

Table 4.4
Inference Results of ShuffleNetV2x1.0 Characteristics and Performances on Desktop Computer

Sparsity Level (%)	Number of Parameters	FLOPs	Model Size (MB)	Accuracy (%)	Inference Time, CPU (ms)	Inference Time, GPU (ms)
0	1.26M	151.69M	4.87	96.74	26.61	15.02
10	1.13M	25.13M	4.32	96.49	24.84	14.7
20	1.01M	22.79M	3.86	95.84	27.03	14.54
25	945K	21.63M	3.62	95.47	24	14.82
30	**883K**	**20.47M**	**3.38**	**95.19**	**24.66**	**14.83**
40	761K	18.12M	2.92	88.25	24.2	14.87
50	638K	15.8M	2.44	70.09	23.02	14.85
60	515K	13.48M	1.97	46.35	22.37	14.74
70	394K	11.14M	1.51	16.69	21.5	14.84
80	269K	8.82M	1.03	16.69	20.63	14.75
90	148K	6.48M	0.57	16.69	**19.81**	**14.83**

Aside from that, CPU inference time decreased significantly as pruning levels rose. This inference time behaviour can be seen as shown in Figure 4.10. For instance,

the inference time declined from 26.61 ms to less than 20 ms at the highest sparsity levels, which only left the model with 147K parameters. This indicates the effectiveness of channel removal for CPU-based inference runtime. GPU inference time, on the other hand, remained stable at 14 ms to 15 ms regardless of sparsity level. Thus, while the CPU-based inference succeeded, this further supports the fact that GPU throughput was having issues bound by its application programming interface (i.e., API) in order to optimise its kernel.

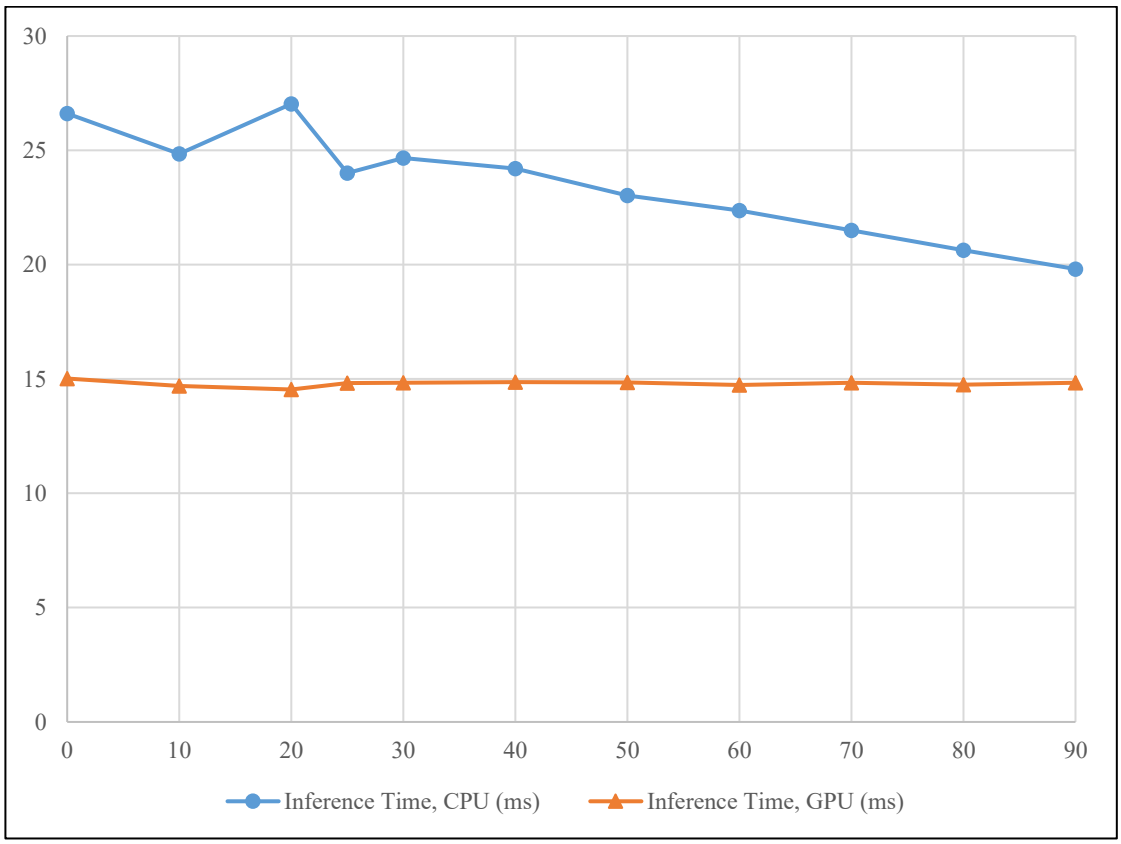

Figure 4.10 Sparsity Level (%) vs. Inference Time on CPU and GPU of ShuffleNetV2x1.0

On top of the other variants, ShuffleNetV2x1.5 had the best baseline performance, which attained 99.43% of classification accuracy without using the optimisation strategies, according to Table 4.5. Moreover, by applying a modest percentage of removed filters up to 25% of sparsity levels, the model can preserve its accuracy over 98% while considerably decreasing the number of parameters from 2.48M to 1.86M and 305.81M to 32.72M FLOPs as well as its model size from 9.57MB to 7.13MB. In spite of that, accuracy loss became apparent after 30% of the sparsity

level, with a progressive reduction to 93.72% at 40% channel elimination, followed by a sharp collapse after 50%.

Table 4.5
Inference Results of ShuffleNetV2x1.5 Characteristics and Performances on Desktop Computer

Sparsity Level (%)	Number of Parameters	FLOPs	Model Size (MB)	Accuracy (%)	Inference Time, CPU (ms)	Inference Time, GPU (ms)
0	2.48M	305.81M	9.57	99.43	29.78	13.87
10	2.23M	38.03M	8.53	99.27	28.57	14.88
20	1.99M	34.48M	7.60	98.9	27.53	15.02
25	1.86M	32.72M	7.13	98.37	27.09	14.74
30	**1.74M**	**30.96M**	**6.66**	**96.49**	**26.3**	**14.92**
40	1.5M	27.41M	5.74	93.72	25.15	14.87
50	1.26M	23.89M	4.80	73.36	23.9	14.97
60	1.01M	20.37M	3.87	47.9	23.85	15.03
70	769K	16.82M	2.95	16.69	22.68	14.8
80	523K	13.30M	2.00	16.69	21.22	14.86
90	281K	9.74M	1.08	16.69	**19.98**	**14.78**

Besides that, Figure 4.11 exhibits the comparison of inference runtime between the CPU-based and GPU-based performances. It can be seen that CPU inference time gradually became faster as the sparsity levels increased, speeding up from 29.78 ms to less than 20 ms at aggressive optimisation settings, which reduced the number of parameters to 281K with 9.74M FLOPs and 1.08 MB of the model size. In contrast, GPU inference time was virtually constant, varying approximately between 14 ms and 15 ms throughout all the optimisation levels. Therefore, this behaviour demonstrates that for bigger ShuffleNetV2 variants, this optimisation strategy produced significant improvements for CPU inference while providing minimal increases for GPU inference.

Last but not least, the ShuffleNetV2x2.0 results tabulated in Table 4.6 display the largest starting capacity at 5.36M number of parameters, computational cost with 596.19M FLOPs, and model size with 20.57MB due to its width multiplication features. Here, classification accuracy was impressively robust up to 25% trimming of sparsity level that remained steadily over 98% and exceeded 96% even at 30% of channel removal. Nevertheless, beyond this threshold, the accuracy losses increased similarly to the previous variants. Although in comparison, the model yielded and maintained higher

accuracy than the smaller variants at the same sparsity level, this is because of its higher starting capacity and redundancy of zeroed weights.

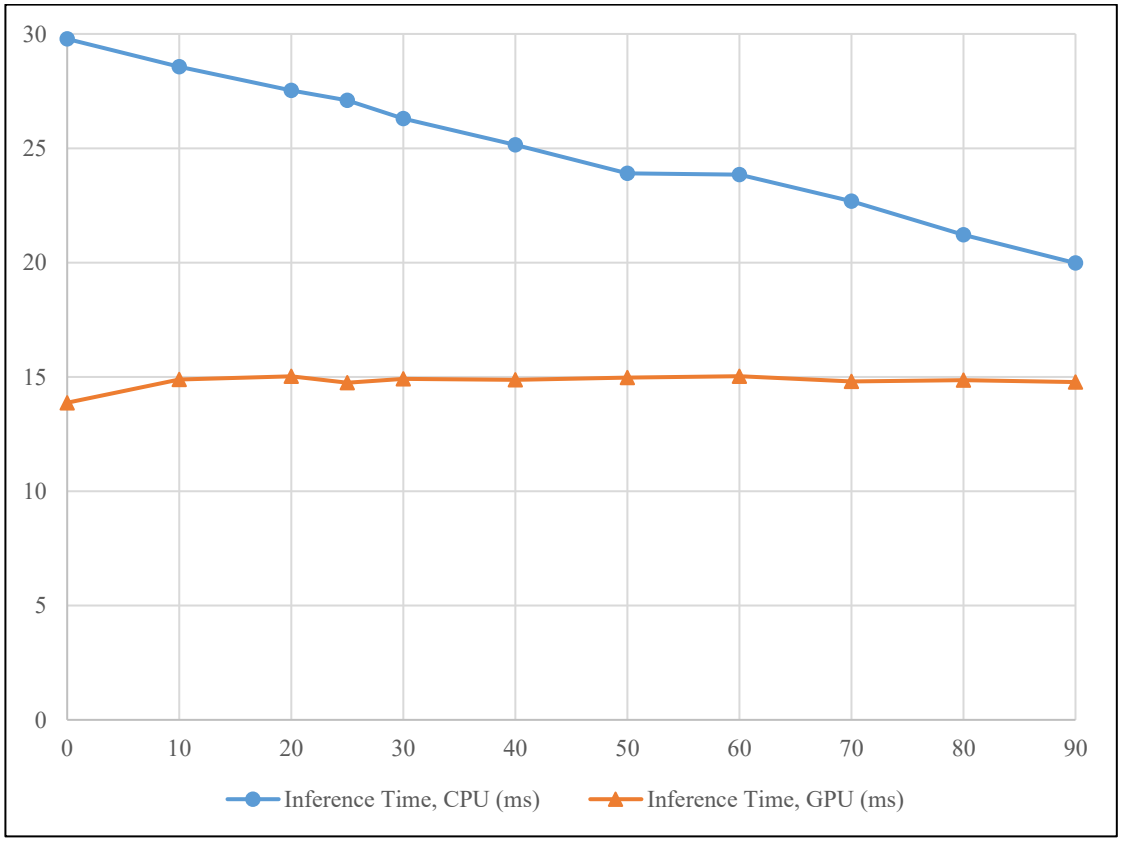

Figure 4.11 Sparsity Level (%) vs. Inference Time on CPU and GPU of ShuffleNetV2x1.5

Table 4.6
Inference Results of ShuffleNetV2x2.0 Characteristics and Performances on Desktop Computer

Sparsity Level (%)	Number of Parameters	FLOPs	Model Size (MB)	Accuracy (%)	Inference Time, CPU (ms)	Inference Time, GPU (ms)
0	5.36M	596.19M	20.57	98.37	50.57	15.75
10	4.81M	96.71M	18.39	98.29	51.57	16.78
20	4.28M	86.91M	16.36	98.41	44.61	15.15
25	4.02M	82.03M	15.36	98.04	41.69	16.24
30	**3.76M**	**77.15M**	**14.36**	**96.04**	**41.21**	**16.33**
40	3.22M	67.35M	12.32	91.92	39.09	16.6
50	2.7M	57.54M	10.31	76.91	37.77	16.45
60	2.17M	47.74M	8.29	65.65	32.37	14.5
70	1.64M	37.93M	6.25	31.37	31.47	14.82
80	1.11M	28.17M	4.25	40.07	29.15	14.67
90	580K	18.37M	2.22	16.52	**28.35**	**14.63**

Also, the most noticeable effect of channel removal can be seen clearly, especially on CPU-runtime inference performance. Figure 4.12 further depicts the response between the CPU and GPU performance. With that, by the original model itself, as in 0% of sparsity level, CPU inference took more than 50 ms, which made it much slower than the inference time on the GPU. However, incremental channel reduction reduced CPU inference time by roughly half, which reached around 28 ms at 90% channel loss. Yet GPU inference time remained rather insensitive to these optimisation strategies and stayed constantly around 14 ms to 16 ms in most of the sparsity levels.

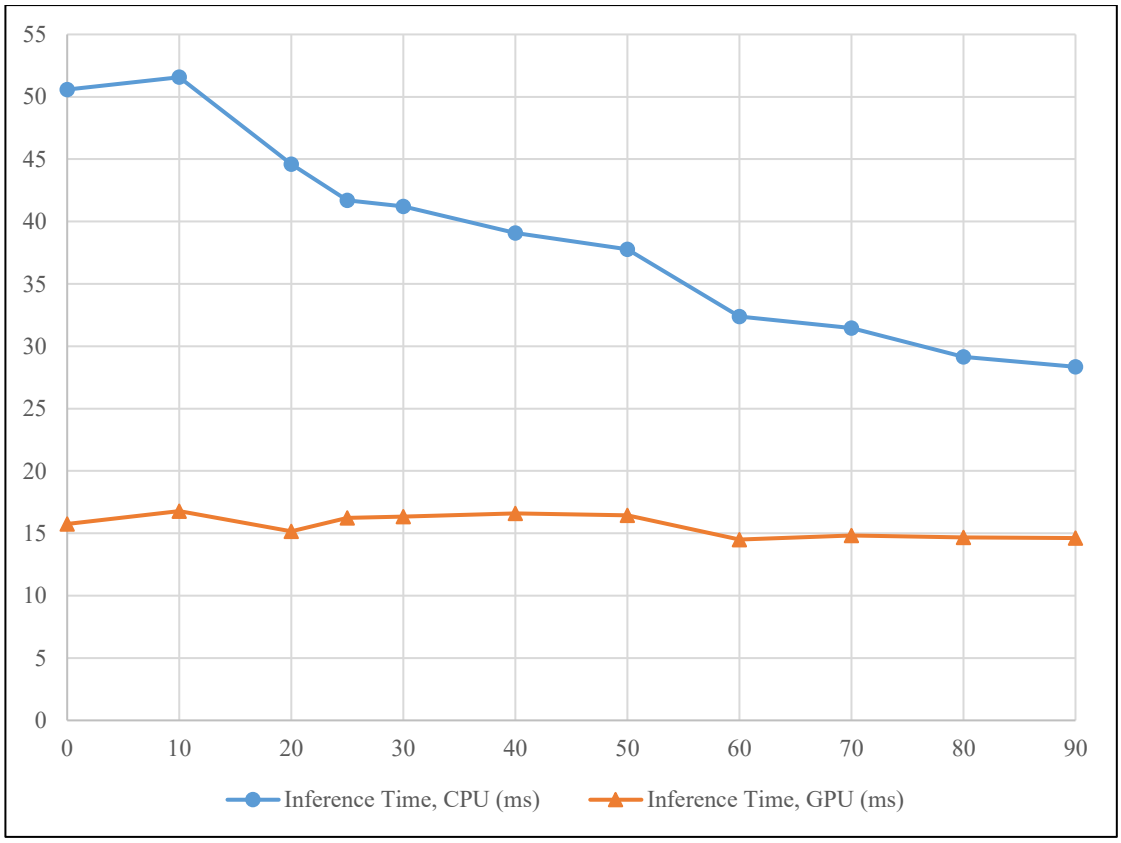

Figure 4.12 Sparsity Level (%) vs. Inference Time on CPU and GPU of ShuffleNetV2x2.0

In short, the explanation above shows that the strategy of post-train structured pruning with simplification is a good optimisation method for increasing inference efficiency of the ShuffleNetV2 model on desktop computer platforms, especially for CPU-based runtimes. This strategy not only decreases model size and computational complexity, but it may also maintain or even improve accuracy by removing duplicate feature channels. Moreover, bigger ShuffleNetV2 variants can have higher accuracy

across the same sparsity levels, which demonstrates that over-parameterised models have more redundancy. Nevertheless, there is a need to carefully select the sparsity levels, as excessive channel reduction degrades accuracy. Furthermore, the minimal impact on GPU inference time implies that additional optimisation strategies are required to fully utilise this lightweight model. Hence, this analysis has revealed several fundamental differences in the behaviour of CPU-based runtime and GPU-based runtime performances with respect to the applied optimisation strategies.

4.3.2 Comparative Analysis between the Original Model and the Optimised Models

In conjunction with the above explanation, a comparative analysis between the variants of the original model and the optimised model of ShuffleNetV2 can further provide an additional analysis from a different perspective that can assist in further justifying the optimisation strategies. Across all the sparsity levels, the moderate channel reduction of 30% is chosen as the most optimised performance that has consistently achieved the best balance of accuracy and efficiency across all variants. Therefore, Table 4.7 compiles and summarises the data of the selected sparsity level with all of its obtained results.

Table 4.7
Performance Comparison of ShuffleNetV2 Variants: Original vs. Optimised Model

Variants	**x0.5**		**x1.0**		**x1.5**		**x2.0**	
Model	**Ori.**	**Optim.**	**Ori.**	**Optim.**	**Ori.**	**Optim.**	**Ori.**	**Optim.**
Num. of param.	348K	**244K**	1.26M	883K	2.48M	1.74M	5.36M	3.76M
FLOPs	43.55M	**8.57M**	151.69M	20.47M	305.81M	30.96M	596.19M	77.15M
Model Size (MB)	1.36	**0.94**	4.87	3.38	9.57	6.66	20.57	14.36
Accuracy (%)	98.49	**96.74**	96.74	95.19	99.43	96.49	98.37	96.04
Inf. Time, CPU (ms)	15.5	**14.4**	26.61	24.66	29.78	26.3	50.57	41.21
Inf. Time, GPU (ms)	13.91	15.02	15.02	14.83	13.87	14.92	15.75	16.33

First and foremost, all ShuffleNetV2 variations showed considerable reductions in parameters and FLOPs in the context of computational complexity and capacity. The

x0.5 variants had achieved the greatest decrease, with the number of parameters dropping from 347K to 244K and FLOPs dropping from 43.55M to 8.57M, which resulted in a roughly 80% reduction in computing cost. Similarly, the x1.0, x1.5, and x2.0 variants also had significant absolute FLOPs reductions, which reduced from 151.69M to 20.47M, 305.81M to 30.96M, and 596.19M to 77.15M, respectively. Consequently, with the reduction of computational complexity, such as the number of parameters, it contributed to the reduction of capacity, as in model size. Thus, it can be seen that from x0.5 to x2.0 variants, all models' sizes dropped from 1.36 MB to 0.94 MB, 4.87 MB to 3.38 MB, 9.57 MB to 6.66 MB, and 20.57 MB to 14.36 MB. This finding emphasises that the optimisation strategies can increase the lightness of this lightweight model.

Furthermore, in terms of classification accuracy, the optimised models expressed a good and comparable performance across all variations, even though there was a slight loss of around 1% to 3% in accuracy when compared to the original design. Specifically, the x0.5 variant's accuracy reduced significantly from 98.49% to 96.74%, while the x1.0 variant's loss was less, from 96.74% to 95.19%. Likewise, the x1.5 and x2.0 variations follow similar patterns, with accuracy maintaining over 96% following optimisation in contrast to their original accuracy at 99.43% and 98.37%, respectively. Ultimately, despite substantial reductions in model complexity, all optimised models maintained a reasonable accuracy, and this behaviour implied that a considerable number of parameters in the original networks were redundant and could be deleted without negatively impacting classification performance. In other words, by using the applied optimisation strategies, the ShuffleNetV2 model as a whole can sustain its robustness at the 30% sparsity level.

Aside from the other characteristics that have been mentioned above, inference time is also one of the crucial criteria that defines a model as lightweight, especially for edge applications that focus on strict latency or real-time requirements. This is because the difference in performance of inference time can clearly demonstrate and straightforwardly show the practical advantages of optimisation between several lightweight models. As such, based on the table, the x2.0 variant achieved the greatest decrease in CPU inference time, which dropped around 9 ms from 50.57 ms to 41.21 ms. Meanwhile, the x0.5, x1.0, and x1.5 variants also saw performance increases but only with small changes in duration, with around 1 ms, 2 ms, and 3 ms, respectively

(e.g., from 15.5 ms to 14.4 ms, from 26.61 to 24.66 ms, and from 29.78 ms to 26.3 ms, respectively).

As opposed to the CPU-based inference time, the GPU-based inference times were shown to be remarkably consistent between original and optimised models across all of the variations, with only slight differences that can be observed. Moreover, those varied durations also did not present any trends or relationships with respect to the optimisation strategies or any of the resulting characteristics. For instance, the x0.5 variants yielded a bit slower time, from 13.91 ms to 15.02 ms; however, the x1.0 variants acquired a slightly faster time, from 15.02 ms to 14.83 ms. Yet the x1.5 and x2.0 variants became a little slower again, from 13.87 to 14.92 ms and 15.75 ms to 16.33 ms, respectively. As such, it can only be concluded that further investigations with more data are required in order to justify this finding.

In a nutshell, this comparative analysis shows many insights and benefits across all the ShuffleNetV2 variants with more details. For the most part, several characteristics, such as computational complexity and model capacity, displayed an improvement in performance. Meanwhile, the classification accuracy also yielded comparable and reasonable results in contrast with the original model. Technically, these characteristics that are applied with the optimisation strategies are subject to model-dependent behaviour. As a result, the majority of the results were as expected based on the methodology of the evaluation metrics. However, the inference time is subject to hardware-dependent behaviour. As such, it can be seen that the CPU-based inference time can contribute to an increase in latency, while the GPU-based inference time did not show any improvement. For now, it can be said that the difference in GPU computation is what hinders the optimisation strategies. Lastly, in terms of the optimised variants comparison, the most lightweight model was the x0.5 variant, which has the lightest characteristics for this test case. Nevertheless, the next subsection will further explore several different image classification tasks that will be deployed on different edge devices in order to proceed with a more detailed analysis.

4.4 Performance Evaluation of the Optimised Lightweight CNN Model on Edge Devices using Appropriate Evaluation Metrics

Based on the findings and interpretations of the previous subsection, several key insights have been highlighted, particularly for the inference performance of

ShuffleNetV2, including its variations in a desktop computer environment. Primarily, most evaluation metrics, such as the number of parameters, FLOPs, model size, and classification accuracy, have been properly clarified. Nonetheless, one crucial metric (i.e., inference time) indicated differently, specifically for models executed by the GPU-based runtime. For that, this subsection is motivated to further investigate the relationship between the optimisation strategies and inference time by examining the results of the ShuffleNetV2 model trained with several different image classification datasets, which were employed in two different edge devices (i.e., the Jetson Nano and the Raspberry Pi 5). Also, the classification accuracy will still be included for discussion, as this quality factor is critical for measuring the models' accuracy-efficiency trade-off. Finally, some results from related studies will be compiled and analysed with the obtained results for a comparative analysis.

4.4.1 Inference Performance Evaluation on Jetson Nano and Raspberry Pi 5

The following results of the inference performance across four datasets and several sparsity levels among two embedded hardware platforms have revealed some persistent and interesting tendencies regarding classification accuracy and inference latency of the deployment efficiency. Specifically, the several datasets that were utilised were categorised according to the size of the dataset and its applications. As such, this research has included a mid-sized dataset (i.e., the classification of groundnut plant leaf diseases and the classification of ball screw drive surfaces), a small-sized dataset (i.e., the classification of material waste, namely as TrashNet), and a big-sized dataset (i.e., the classification of concrete crack surfaces). Moreover, inference performance was evaluated on Jetson Nano (i.e., GPU-based hardware) and Raspberry Pi 5 (i.e., CPU-based hardware) in order to evaluate the practical deployment features of the optimised strategy on resource-constrained devices. With that, this examination will show how channel removal affects accuracy and latency across limited-resource hardware with distinctive image classification applications.

To begin with, Table 4.8 shows the summarised data pertaining to the results of inference performance for groundnut leaf disease classification. For the most part, as the dataset has gone through inference on the desktop computer environment, similarly, it can be seen that the classification accuracy always yielded the same results for any of the sparsity levels of the optimised strategy. On the other hand, Figure 4.13 illustrates

a line graph of the sparsity levels against the inference time of ShuffleNetV2 variations on the Jetson Nano and Raspberry Pi 5 (i.e., denoted as JNx0.5 and RP5x0.5 for ShuffleNetV2x0.5, JNx1.0 and RP5x1.0 for ShuffleNetV2x1.0, JNx1.5 and RP5x1.5 for ShuffleNetV2x1.5, and JNx2.0 and RP5x2.0 for ShuffleNetV2x2.0, respectively). As a whole, it can be seen clearly that the inference time on the Raspberry Pi 5 produced a better performance compared to the Jetson Nano, where it remained rather constant.

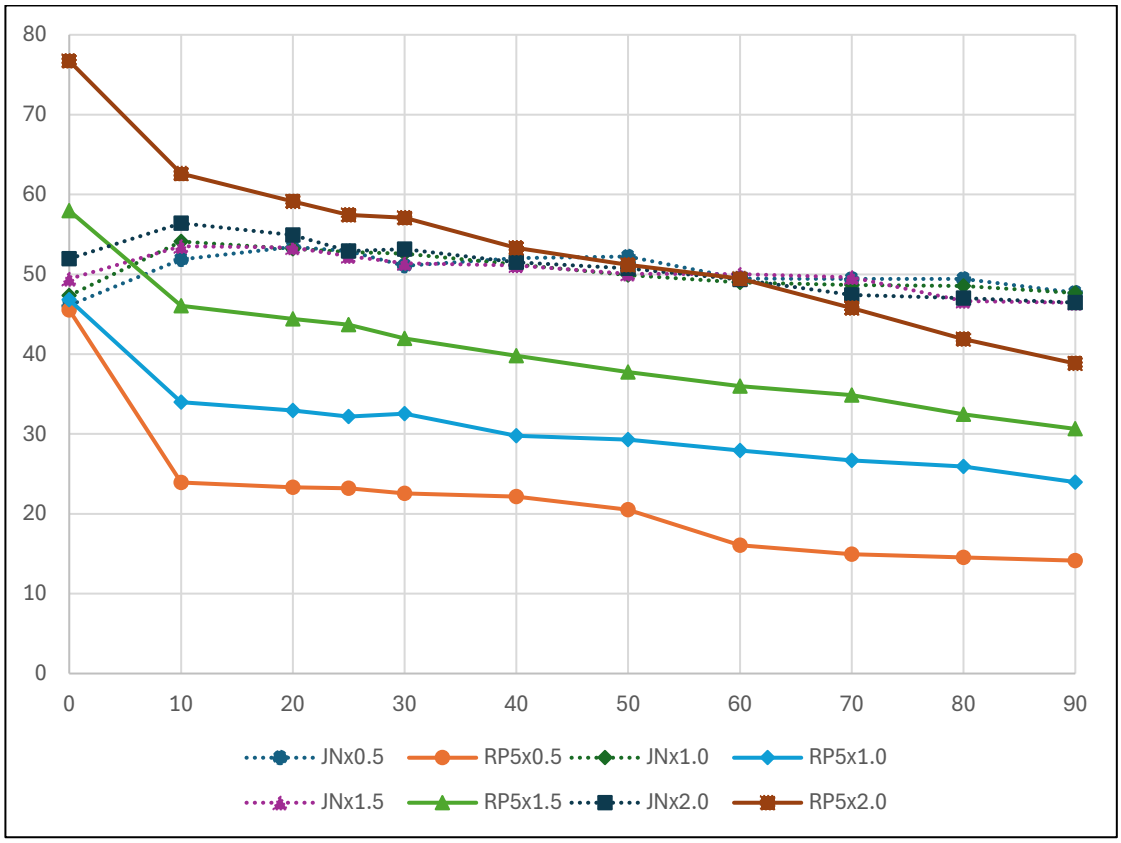

Figure 4.13 Inference Time on CPU and GPU vs. Sparsity Level (%) of the Optimisation on Groundnut Plant Leaf Diseases Classification Task

For example, the inference time of RP5x0.5 decreased by around 50% for the 30% sparsity level, which was from 45.54 ms to 22.55 ms, and further dropped down to 14.14 ms for the 90% sparsity level as opposed to the inference time of JNx0.5. Likewise, other variations also displayed a similar pattern of inference time on the Raspberry Pi 5 against the Jetson Nano. This indicates that channel removal effectively reduced computational demand on the Raspberry Pi 5 while not being susceptible to the Jetson Nano. Besides that, the RP5x0.5 and RP5x1.0 had already lower initial inference times compared to their Jetson Nano counterparts. As opposed to this, the RP5x1.5 and RP5x2.0 had a bit higher initial inference times compared to their Jetson Nano

counterparts. When observed, RP5x1.5 at around 5% sparsity level and RP5x2.0 at around 60% sparsity level marked the intercept points where the transition occurred. Hence, it can be suggested that these seem to be the points where the Raspberry Pi 5 started to perform better than the Jetson Nano.

Next, the second application concerning the ball screw drive defect detection was another mid-sized dataset that was used to examine the resilience and deployment appropriateness by using the proposed optimised strategy. Accuracy-wise, based on Table 4.9, it seems that the optimised sparsity level, which can maintain remarkable accuracy with only a loss of about 1-3% of accuracy, was up to 50% for the x0.5 variant, 40% for the x1.0 and x1.5 variants, and 30% for the x2.0 variant (e.g., from 91.79% to 89.91%, 92.72% to 89%, 92.55% to 91.73%, and 91.83% to 89.99%, respectively). Additionally, in comparison with the first application, a growth in accuracy performance can be observed for the first three variants (i.e., the x0.5, x1.0, and x1.5 variants) as opposed to the first application, which only yielded the optimised sparsity level at 30% for all its variants. For that, this implies that the dataset size can be included as a factor to improve the robustness of the ShuffleNetV2 model, since the second dataset is slightly larger than the first dataset.

Moving on to the inference time performance, Figure 4.14 presents the plotted line graph across all sparsity levels for all ShuffleNetV2 variants, which were deployed on Jetson Nano and Raspberry Pi 5. With that, several relationships were discovered, including that the trends exhibited similar patterns with the previous application in terms of the inference time of Jetson Nano against the inference time on Raspberry Pi 5. Concisely, the inference times across RP5x0.5 through RP5x2.0 expressed shorter times as the sparsity levels increased, while inference times across JNx0.5 through JNx2.0 did not show any significant improvements. In contrast, the obvious difference that can be traced was the initial inference time values on the Jetson Nano, which seem to drop down a bit for this test case. Also, the intercept points where the transition for Raspberry Pi 5 to start performing better than the Jetson Nano had increased slightly, whereby RP5x0.5 and RP5x1.0 were around 3% sparsity level, RP5x1.5 around 8% sparsity level, and RP5x2.0 around 75% sparsity level. So, while the varied dataset may have caused this irregularity to happen, further investigation is required to help provide more clarity on this behaviour.

Next, the third dataset used for the image classification task on edge devices was TrashNet, a small dataset. With that, Table 4.10 depicts the results obtained from the

correlation between the percentage of channel removal and accuracy, as well as inference time, for ShuffleNetV2 variations. Once again, it can be seen in the data obtained that as the sparsity level increased, the accuracy declined significantly after a certain optimised level. For instance, these were true for all variants around 10-20% sparsity levels that managed to sustain the accuracy of more than 81%, which seems to still be comparable, considering the original accuracy. On top of that, the x1.0 variants also managed to break through with a 25% sparsity level, which acquired an accuracy of 82.48%. While the behaviour of lower optimised levels compared to the previous dataset was expected due to the small dataset, the tremendous hyperparameter configurations and data augmentations for this particular model training also contributed to the rising original model accuracy. These distinct responses can actually be noticed, as from the x0.5 to x2.0 variants, the original accuracy rose from 82.68% to 87.6% compared to each previous dataset, where their original model accuracies stayed roughly similar across the variations. Thus, this discovery seems to hint that the advantage of the higher variants can withstand an extensive training strategy for accuracy improvement, unlike the lower variants.

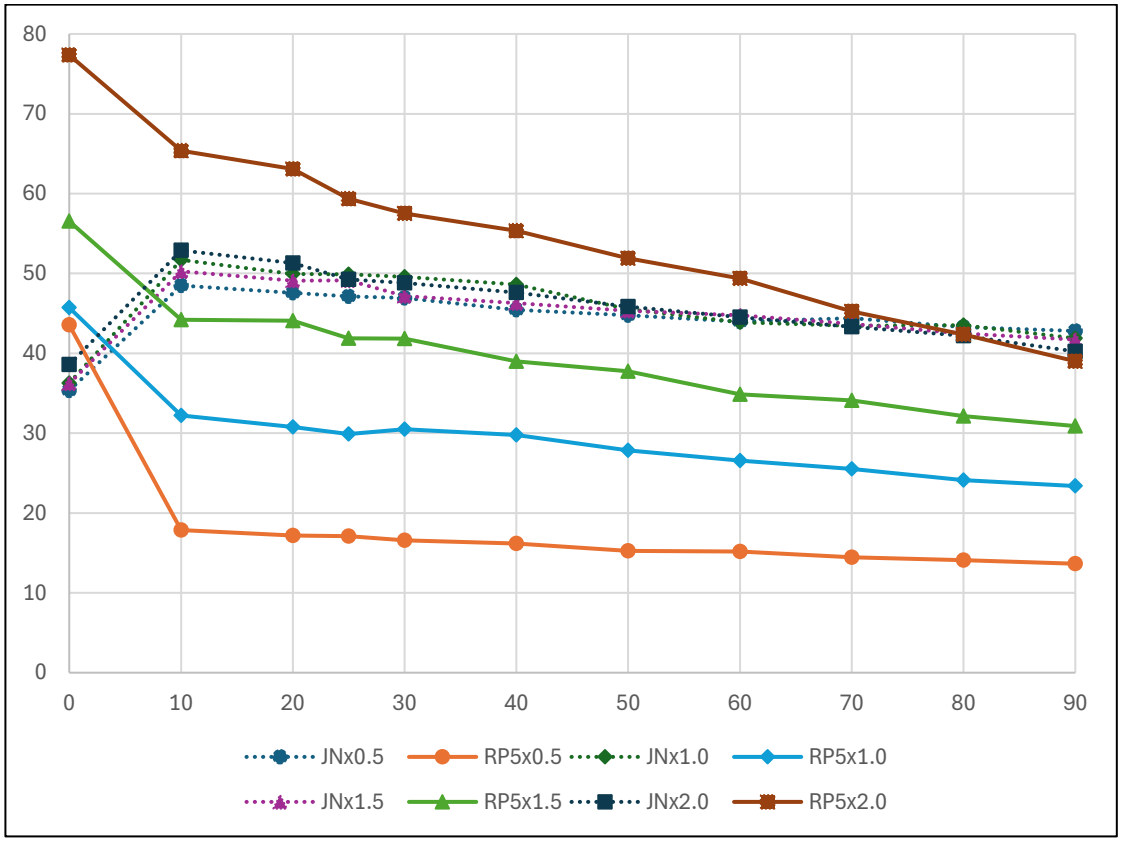

Figure 4.14 Inference Time on CPU and GPU vs. Sparsity Level (%) of the Optimisation on Ball Screw Drive Defect Classification Task

Besides that, again, the inference performance for this particular dataset also emphasised a similar relationship as the previous dataset in terms of the percentage of channel removal against the inference time on Jetson Nano and Raspberry Pi 5, according to Figure 4.15. In addition, the patterns resembled the pattern from the first dataset, where the original starting point for RP5x0.5 and RP5x1.0 was less than JNx0.5 and JNx1.0, and the intercept point of transition, where RP5x1.5 and RP5x2.0 started to perform better than JNx1.5 and JNx2.0, was around 4% and 60% sparsity levels, respectively. Therefore, the connection to relate this characteristic to the dataset size variations may not be valid. On top of that, a similar rapid reduction of inference time from 0% to 10% sparsity levels can be detected for all the variations deployed on Raspberry Pi 5 across all three datasets. On the contrary, a similar small surge of inference time from 0% to 10% sparsity levels can be spotted for the majority of the variations deployed on Jetson Nano across all three datasets. As it may seem, these two distinguished reactions may be related to the same factor that caused them, as the results showed that they opposed each other on the same sparsity level.

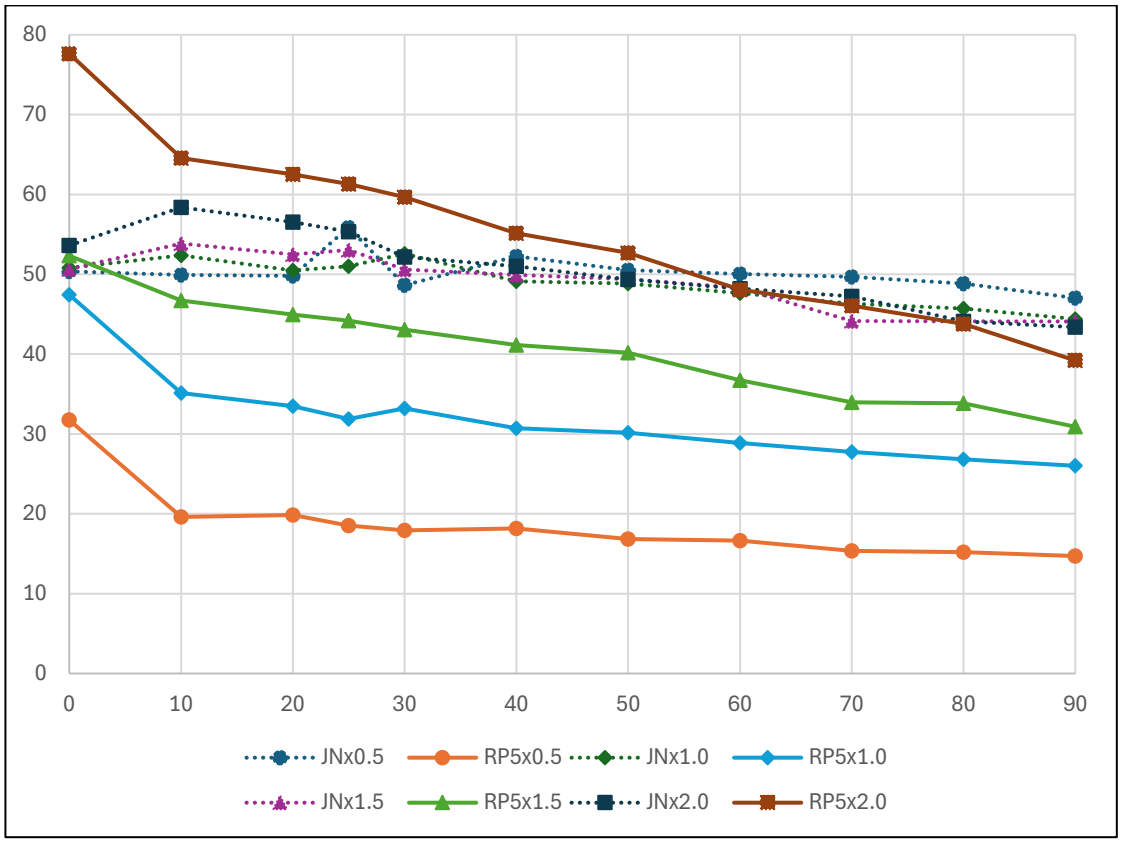

Figure 4.15 Inference Time on CPU and GPU vs. Sparsity Level (%) of the Optimisation on TrashNet Classification Task

Finally, the fourth dataset for the image classification task was the biggest sample used for this research in order to assess one of the generalisation criteria for the optimised strategy. Table 4.11 tabulates the classification results of the concrete surface crack detection application. Regarding that, the impact of a large-scale dataset can be seen clearly, where the optimised sparsity levels were around double the amount of the first two datasets. For example, the x0.5 variant succeeded in achieving a 70% sparsity level while retaining 90.91% accuracy; the x1.0 and x1.5 variants sustained an 80% sparsity level with 91.24% and 90.62% accuracies, while the 2.0 variant maintained a 60% sparsity level with 91.52% accuracy. In particular, even with 90% of channel removal for x1.0 and x1.5 variants, the dropped accuracies only fell to 87.96% and 83.45%, which were still reasonable values compared to the other two variants. Also, this goes to show that the x1.0 and x1.5 managed to keep low-redundant weights during their model training. Hence, with this observation, it just proves that a bigger dataset has a tendency to produce a highly robust model with low-redundant weights, which can provide a higher optimised sparsity level.

Furthermore, on the inference-time side, Figure 4.16 shows the relationships between sparsity levels and inference time on the Jetson Nano and Raspberry Pi 5 for the concrete surface crack image classification task. For the most part, the resulting graph emphasised that the interactions that occurred were closely similar to those observed in the previous image classification tasks. Likewise, a lower starting inference time at 0% of channel removal can be observed on RP5x0.5 with 45.54 ms and RPx1.0 with 46.8 ms as opposed to JNx0.5 with 46.11 ms and JNx1.0 with 47.34 ms. Also, the transition where RPx1.5 and RPx2.0 cross over the JNx1.5 and JN2.0 inference times performances were around 5% and 60% sparsity levels, respectively, which were too similar to the first dataset's behaviour. Additionally, other consistent characteristics, such as the rapid reduction time on Raspberry Pi 5 from 0% to 10% sparsity level, as well as the small fluctuation (i.e., also from 0% to 10% sparsity level) and constant time on Jetson Nano, can still be distinguished based on the figure. Thus, this final dataset provided several enlightenments of the optimised strategy conducted on a lightweight CNN model in resource-constrained devices.

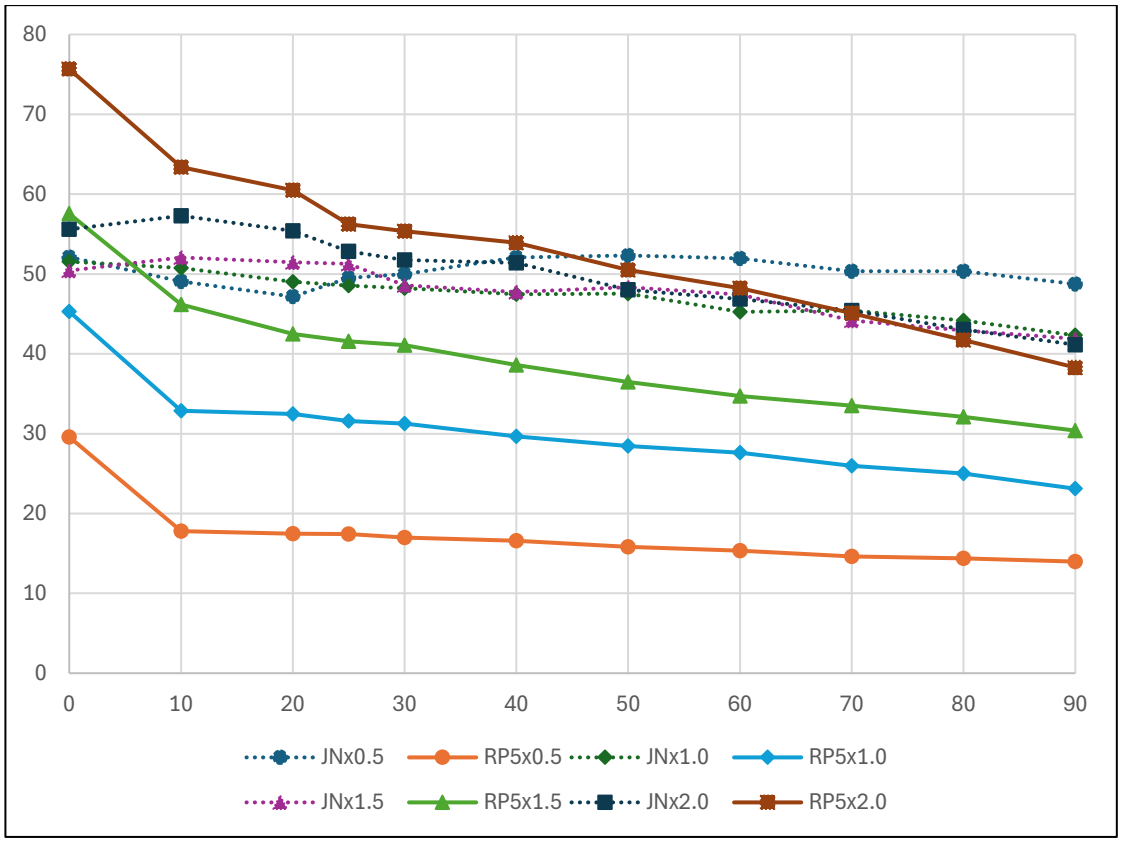

Figure 4.16 Inference Time on CPU and GPU vs. Sparsity Level (%) of the Optimisation on Concrete Surface Crack Classification Task

In a nutshell, the results suggest that the lightweight CNN architecture, particularly ShuffleNetV2 variants, can achieve a great balance between accuracy and real-time performance when carefully matched with dataset features and target hardware. When it comes to deployment, the Raspberry Pi 5 gets significantly more flexibility and feasibility than the Jetson Nano. In depth, the size of datasets surely matters when it comes to increasing the robustness of the model and thus boosting the optimised sparsity level. Additionally, an extensive training method, including hyperparameter configurations and data augmentations, may further raise the accuracy, which highlights the advantage of using the large variant of ShuffleNetV2. Besides that, the inference time performance on Raspberry Pi 5 clearly proves the advantages of the optimised strategy for CPU-based hardware compared to GPU-based hardware (i.e., the Jetson Nano). Nevertheless, as followed by the results from the previous subsection, the current optimised strategy still needs to be improved in order to be utilised for GPU-based hardware. Other than that, the significant reduction of inference time for Raspberry Pi 5 from 0% to 10% sparsity level can be related to the huge drop in FLOPs. This can be referred to in Table 4.11 until Table 4.14, where the huge drop in FLOPs

occurred across all variants from 0% to 10%, while after that, the FLOPs decreased slowly. One thing that can be implied as to why this behaviour cannot be seen earlier is that the desktop computer is highly resource-intensive compared to the Raspberry Pi 5. In short, these results evaluate the significance of optimisation strategies level, training method, and dataset size effects on the inference performance of different limited-resource hardware applications.

Table 4.8
Performance Comparison of ShuffleNetV2 Variants based on Image Classification Task of Groundnut Plant Leaf Diseases Dataset

Sparsity Level (%)	x0.5			x1.0			x1.5			x2.0		
	Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)	
		Jetson Nano	RPi5		Jetson Nano	RPi5		Jetson Nano	RPi5		Jetson Nano	RPi5
0	98.49	46.11	45.54	96.74	47.34	46.8	99.43	49.41	57.97	98.37	51.97	76.73
10	98.49	51.87	23.92	96.49	54.11	33.99	99.27	53.52	46.06	98.29	56.4	62.6
20	98.57	53.43	23.32	95.84	53.21	32.95	98.9	53.35	44.43	98.41	54.92	59.12
25	98.65	52.91	23.19	95.47	52.78	32.19	98.37	52.29	43.68	98.04	52.92	57.45
30	**96.74**	**51.08**	**22.55**	**95.19**	**52.59**	**32.55**	**96.49**	**51.38**	**41.96**	**96.04**	**53.14**	**57.09**
40	83.11	52.02	22.14	88.25	51.21	29.77	93.72	51.12	39.82	91.92	51.5	53.32
50	58.22	52.22	20.51	70.09	49.92	29.3	73.36	50.05	37.74	76.91	50.73	51.18
60	16.73	49.47	16.05	46.35	48.98	27.93	47.9	50.02	36.01	65.65	49.34	49.46
70	16.69	49.42	14.94	16.69	48.68	26.69	16.69	49.62	34.86	31.37	47.43	45.78
80	16.69	49.42	14.54	16.69	48.56	25.94	16.69	46.64	32.48	40.07	47.01	41.89
90	16.69	47.68	14.14	16.69	47.64	23.97	16.69	46.43	30.64	16.52	46.46	38.83

Table 4.9
Performance Comparison of ShuffleNetV2 Variants based on Image Classification Task of Ball Screw Drive Dataset

Sparsity Level (%)	**x0.5**			**x1.0**			**x1.5**			**x2.0**		
	Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)	
		Jetson Nano	RPi5		Jetson Nano	RPi5		Jetson Nano	RPi5		Jetson Nano	RPi5
0	91.79	35.34	43.55	92.72	36.28	45.72	92.55	36.21	56.54	91.83	38.58	77.36
10	91.74	48.49	17.87	92.67	51.7	32.23	92.93	50.27	44.2	91.94	52.89	65.36
20	91.77	47.58	17.2	92.41	49.92	30.78	92.78	49.09	44.09	91.59	51.3	63.09
25	91.76	47.15	17.12	92.03	49.89	29.9	92.61	49.15	41.88	91.1	49.24	59.37
30	91.64	46.92	16.57	91.5	49.59	30.48	92.38	47.18	41.86	**89.99**	**48.81**	**57.5**
40	91.15	45.44	16.17	**89**	**48.63**	**29.79**	**91.73**	**46.3**	**38.99**	80.43	47.6	55.35
50	**89.91**	**44.75**	**15.26**	82.04	45.52	27.86	85.68	45.32	37.77	62.1	45.86	51.92
60	77.73	43.95	15.18	52.06	43.9	26.57	71.31	44.77	34.87	50.72	44.52	49.36
70	51.33	44.42	14.46	50.72	43.44	25.51	53.42	43.67	34.12	50.72	43.33	45.24
80	50.67	43.27	14.09	50.72	43.51	24.1	49.3	42.49	32.12	50.72	42.21	42.38
90	49.28	42.81	13.65	50.72	41.88	23.38	50.72	41.71	30.89	50.72	40.25	39.01

Table 4.10
Performance Comparison of ShuffleNetV2 Variants based on Image Classification Task of TrashNet Dataset

Sparsity Level (%)	x0.5			x1.0			x1.5			x2.0		
	Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)	
		Jetson Nano	RPi5		Jetson Nano	RPi5		Jetson Nano	RPi5		Jetson Nano	RPi5
0	**82.68**	50.38	31.75	**84.45**	50.79	47.44	**84.45**	50.66	52.32	**87.6**	53.59	77.64
10	82.87	49.91	19.59	84.45	52.38	35.12	**85.43**	**53.85**	**46.71**	86.42	58.38	64.55
20	**81.5**	**49.8**	**19.85**	83.86	50.5	33.49	78.54	52.48	44.94	**83.86**	**56.54**	**62.52**
25	78.54	55.86	18.52	**82.48**	**50.99**	**31.86**	71.65	53.06	44.18	79.53	55.33	61.31
30	74.61	48.62	17.92	74.21	52.57	33.21	64.75	50.62	43.07	73.62	52.18	59.65
40	49.41	52.23	18.16	37.8	49.13	30.72	34.84	49.91	41.12	57.28	51	55.14
50	27.17	50.52	16.86	23.43	48.82	30.14	24.8	49.39	40.19	25.98	49.37	52.67
60	19.88	50.03	16.65	23.43	47.62	28.88	23.43	48.26	36.74	14.76	48.18	48.08
70	19.88	49.67	15.34	24.8	46.33	27.75	23.43	44.15	33.95	19.09	47.22	46.08
80	19.09	48.82	15.2	23.62	45.71	26.81	23.43	44.1	33.83	19.09	44.12	43.77
90	19.09	47.03	14.71	8.86	44.4	26.01	23.43	44.09	30.9	23.43	43.37	39.2

Table 4.11
Performance Comparison of ShuffleNetV2 Variants based on Image Classification Task of Concrete Surface Crack Dataset

Sparsity Level (%)	x0.5			x1.0			x1.5			x2.0		
	Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)		Acc. (%)	Inf. Time (ms)	
		Jetson Nano	RPi5		Jetson Nano	RPi5		Jetson Nano	RPi5		Jetson Nano	RPi5
0	91.41	52.1	29.59	91.92	51.55	45.3	92.14	50.42	57.55	92.32	55.59	75.67
10	91.41	49.06	17.81	91.92	50.77	32.86	92.14	52.07	46.17	92.31	57.29	63.4
20	91.41	47.17	17.46	91.92	49.02	32.48	92.15	51.47	42.51	92.32	55.41	60.5
25	91.41	49.51	17.44	91.93	48.55	31.59	92.15	51.29	41.55	92.33	52.84	56.23
30	91.41	49.98	17	91.93	48.22	31.28	92.15	48.6	41.1	92.33	51.74	55.38
40	91.41	52.06	16.57	91.91	47.45	29.67	92.16	47.76	38.59	92.34	51.4	53.91
50	91.38	52.32	15.81	91.93	47.55	28.44	92.16	48.35	36.47	92.37	48.04	50.53
60	91.43	51.96	15.34	91.89	45.27	27.62	92.12	47.42	34.71	**91.52**	**46.86**	**48.22**
70	**90.91**	**50.35**	**14.62**	91.71	45.41	25.97	92.03	44.16	33.51	29.65	45.43	45.11
80	29.65	50.34	14.37	**91.24**	**44.19**	**25.01**	**90.62**	**42.93**	**32.11**	29.65	43.05	41.74
90	29.65	48.73	13.98	**87.96**	42.32	23.11	**83.45**	41.89	30.39	29.65	41.14	38.29

4.4.2 Comparative Analysis in terms of Classification Accuracy and Inference Time

Following the previous results, the proposed optimised ShuffleNetV2 model was compared to related research on image classification tasks for ball screw drive defect detection, material waste image classification (i.e., TrashNet), and concrete surface crack detection datasets to assess its competitiveness in terms of inference performance. Specifically, the comparison focuses on variations in the applied CNN design, optimisation method, hardware platform, and classification accuracy. As for inference time, however, those previous works did not specify any data. Therefore, Table 4.12 tabulates the summarised data for this comparative analysis.

Table 4.12
Performance Comparison of ShuffleNetV2 Variants: Original vs. Optimised Model

Dataset	Works	CNN Model	Method	Device	Acc. (%)	Inf. Time (ms)
TrashNet classification	(Qin et al., 2021)	MobileNetV2	SoftMax layer optimisation	16GB RAM + NVIDIA RTX 2060 GPU 6GB VRAM	82.48	not stated
			SVM optimization		83.46	not stated
	this study	ShuffleNetV2 x1.0	proposed optimised strategy	Raspberry Pi 5	83.86	33.49
Ball screw drive defect classification	(Rani et al., 2023)	CondenseNet V2	data augmentation, hyperparameter fine-tune, transfer learning	NVIDIA Jetson Xavier NX	98.08	not stated
	this study	ShuffleNetV2 x0.5	proposed optimised strategy	Raspberry Pi 5	91.64	16.57
Concrete surface crack classification	(B. Wang et al., 2021)	VGG16	layer pruning and parameter reduction	Raspberry Pi 3B+	92.40	not stated
	this study	ShuffleNetV2 x1.0	proposed optimised strategy	Raspberry Pi 5	91.24	25.01

Primarily, the ball screw drive defect detection in (Rani et al., 2023) used CondenseNetV2 paired with data augmentation, hyperparameter fine-tuning, and transfer learning to achieve 98.08% accuracy on an NVIDIA Jetson Xavier NX. While this accuracy exceeds the 91.64% attained by the ShuffleNetV2x0.5 on Raspberry Pi 5 of this research, the comparison must take into account both the architectural complexity of CondenseNetV2 and the model training optimisation advantage. As highlighted in the previous subsection in the TrashNet analysis, data augmentation and

hyperparameter optimisation can help yield a better performance in accuracy. In a sense, these combinations, together with transfer learning, surely can improve classification performance. In contrast, the advantage that can be emphasised for this research was the real-time inference in 16.57 ms on Raspberry Pi 5, which suggests excellent appropriateness for industrial edge applications despite a slight loss of accuracy.

Next, one of the previous research studies on the TrashNet dataset used MobileNetV2 with SoftMax layer optimisation and SVM optimisation on a high-performance workstation with 16 GB RAM and an NVIDIA RTX 2060 GPU 6 GB (Qin et al., 2021). On top of that, the stated accuracies were 82.48% and 83.46%, with no inference time specified. In comparison, this research's best-performing results on TrashNet obtained 83.86% accuracy using an improved ShuffleNetV2x1.0 running on the Raspberry Pi 5. Despite hardware constraints, the proposed solution outperforms the SoftMax-optimised MobileNetV2 and significantly improves upon the SVM-based configuration while achieving a measured inference time of 33.49 ms. Thus, this illustrates that the suggested optimisation technique may achieve competitive accuracy while providing real-time inference on resource-constrained hardware.

Last but not least, another related study on the concrete surface crack image classification task used VGG16 with layer pruning and parameter reduction on a Raspberry Pi 3B+ and achieved 92.40% classification accuracy (B. Wang et al., 2021). On the other hand, the proposed optimised strategy of this research on ShuffleNetV2x1.0 acquired the best-performing results with 91.24% accuracy on Raspberry Pi 5 with 25.01 ms inference time. While the objective of the method might seem similar in reducing the number of parameters, the method in (B. Wang et al., 2021) focused on a different approach, where it reduced the number of parameters to suit the dataset used, compared to the post-training pruning and simplification used in this research. Also, although the accuracy of this research was a bit lower, it was still comparable, as ShuffleNetV2x1.0 was intrinsically more lightweight and computationally efficient with 77K parameters than the reported VGG16 with 6.82M parameters. Hence, the ShuffleNetV2 advantages increase the feasibility and viability of real-time edge deployment using the proposed optimised strategy.

In conclusion, the findings show that the proposed optimisation technique strikes a good compromise between accuracy and inference efficiency, especially for a CPU-based edge device such as the Raspberry Pi 5. Although some previous studies obtained greater peak accuracy by utilising more powerful GPUs or heavier

architectures, the proposed optimised ShuffleNetV2, however, achieved consistent and competitive performance while drastically decreasing the computational costs. Therefore, this demonstrates the practical benefit of the proposed optimised strategy for real-time, low-power embedded vision systems where efficiency, scalability, and deployment feasibility are essential factors.

4.5 Conclusion

As a conclusion, the results related to the proposed optimisation strategy applied to a lightweight convolutional neural network architecture for the image classification task on edge devices were reported in this chapter. In brief, the results of transfer learning and without transfer learning that were imposed on several lightweight CNN models were analysed to identify the current best-performing model. For that, ShuffleNetV2 achieved the most balanced performance on both transfer learning and without transfer learning conditions (e.g., ShuffleNetV2x0.5 achieved 98.06% accuracy with 9.41ms inference time, while ShuffleNetV2x2.0 yielded 98.64% accuracy with 17.22ms inference time respectively). Following that, the results of the optimised model in terms of the number of parameters, FLOPs, model size, classification accuracy, and inference time were evaluated on a desktop computer platform to prove the effectiveness of the optimisation strategy before deploying the optimised model on edge devices. Specifically, at 30% sparsity level, ShuffleNetV2 maintained comparable accuracy (i.e., above 95%) while reducing parameters by up to 85% and FLOPs by up to 80%. Meanwhile, the lowest inference time can be observed on ShuffleNetV2x0.5 with 14.4ms for CPU-based inference. Finally, the results of inference performance on the selected edge devices, such as the Jetson Nano and Raspberry Pi 5, were discussed as the evaluation results in justifying the performance of the applied optimisation strategy for edge applications across multiple image classification tasks. For example, the model sustained comparable accuracy at approximately 20% sparsity level with 83.86% accuracy and 33.49 ms inference time on the small-scale dataset (i.e., the TrashNet dataset), 30% sparsity level with 91.64% accuracy and 16.57 ms inference time on the medium-scale dataset (i.e., the ball screw drive dataset), and up to 80% sparsity level with 91.24% accuracy and 25.01 ms inference time on the large-scale dataset (i.e., the concrete surface crack dataset) for the optimised model deployed in the Raspberry Pi 5. As opposed to that, on Jetson Nano, inference time remained relatively

constant, revealing hardware-dependent behaviour. Hence, all the results obtained were analysed and explained to achieve the objectives of this research.

CHAPTER 5
CONCLUSION

5.1 Conclusion

In conclusion, this research aimed to improve and optimise a lightweight convolutional neural network architecture tailored specifically for deployment on edge devices for image classification tasks. With that, three objectives were constructed in this research as a strategic way to fulfil the primary aim.

Initially, the first objective to investigate the current lightweight CNN models that are best for the image classification task on edge devices was obtained through a few processes. First of all, it was crucial to understand some background knowledge revolving around lightweight CNN models that are specifically used for the image classification task for edge computing. After that, the evaluation based on transfer learning and without transfer learning was carried out for several selected models, such as MobileNetV2, MobileNetV3, ShuffleNetV2, and VGG16 models, in order to observe their performance comprehensively. By then, the best-performing model, ShuffleNetV2, was identified for having the most consistent performance throughout all the criteria that were discussed, including the training, validation, and inference performance. As a result, ShuffleNetV2 achieved the most balanced performance on both transfer learning and without transfer learning conditions (e.g., ShuffleNetV2x0.5 achieved 98.06% accuracy with 9.41ms inference time, while ShuffleNetV2x2.0 yielded 98.64% accuracy with 17.22ms inference time respectively).

Next, as the second objective, which was to improve and optimise the performance of the identified lightweight CNN architecture according to several appropriate evaluation metrics, was achieved using several continuous experiments. For that, two architecture optimisation methods for improving and optimising the identified model, including post-training pruning and simplification, were constructed and combined to create the effective optimisation strategy. Furthermore, the optimised model that was improved by the optimisation strategy was validated with various related evaluation metrics such as the number of parameters, FLOPs, model size, classification accuracy, and inference time on a desktop computer platform. For example, at 30% sparsity level, ShuffleNetV2 maintained comparable accuracy (i.e., above 95%) while

reducing parameters by up to 85% and FLOPs by up to 80%. Meanwhile, the lowest inference time can be observed on ShuffleNetV2x0.5 with 14.4ms for CPU-based inference Thus, the results proved the effectiveness of the optimised ShuffleNetV2 model before deploying it on edge devices.

Finally, the third objective, which was to evaluate the performance of the optimised lightweight CNN model in classification accuracy and inference time, was achieved by utilising various image classification tasks for edge applications. Here, the ShuffleNetV2 model trained with different image classification datasets was enhanced with the optimisation strategy and employed on the NVIDIA Jetson Nano and the Raspberry Pi 5 to further evaluate the generalisation capacity, together with a short comparison with the related works. Subsequently, the result observed a comparable accuracy and feasible inference time as the lightness (i.e., the sparsity level) increased to a certain extent, which differed for each of the applications. For instance, the model sustained comparable accuracy at approximately 20% sparsity level with 83.86% accuracy and 33.49 ms inference time on the small-scale dataset (i.e., the TrashNet dataset), 30% sparsity level with 91.64% accuracy and 16.57 ms inference time on the medium-scale dataset (i.e., the ball screw drive dataset), and up to 80% sparsity level with 91.24% accuracy and 25.01 ms inference time on the large-scale dataset (i.e., the concrete surface crack dataset) for the optimised model deployed in the Raspberry Pi 5. To sum up, the optimisation strategy was able to optimise the ShuffleNetV2 model for all metrics on the Raspberry Pi 5, while the Jetson Nano was only restricted in terms of inference time.

5.2 Research Contribution

This research contributed to the field of lightweight deep learning for edge computing from both theoretical and practical perspectives. From a theoretical perspective, this study established a comprehensive benchmarking framework for evaluating lightweight CNN architectures under consistent experimental conditions. Through a systematic comparison of multiple lightweight CNN models, the study provided empirical evidence that ShuffleNetV2 offered the best balance between classification accuracy and computational efficiency among the investigated architectures. In addition, the findings demonstrated that the optimal sparsity level depended on dataset characteristics, indicating that different datasets required different

pruning ratios to achieve the best trade-off between accuracy and computational performance.

From a practical perspective, this research proposed and validated an optimisation framework that combined L1-norm structured channel pruning with network simplification to reduce computational complexity while maintaining classification accuracy. The framework was successfully evaluated on both the Raspberry Pi 5 and NVIDIA Jetson Nano, demonstrating its applicability across different edge computing platforms. Furthermore, the study provided practical guidelines for selecting appropriate sparsity levels based on dataset characteristics, which could assist future researchers and practitioners in deploying lightweight CNN models efficiently on resource-constrained edge devices.

5.3 Future Work

Based on the findings of this research, several improvements can be proposed to further enhance the optimisation strategy as future work.

One key recommendation is to address the rapid and significant accuracy degradation observed as the sparsity level increases. This issue can be mitigated by employing iterative pruning, also known as pruning-aware training. Unlike post-train pruning, iterative pruning gradually removes parameters during training according to a defined pruning schedule or frequency. This allows the network to adapt and recover important features or weights after each pruning step and thus stabilises the accuracy from declining greatly at higher sparsity levels.

Another important aspect to improve is the GPU-based inference time limitation, which affects the feasibility and generalisation of the proposed optimisation strategy. However, addressing this issue requires a more comprehensive study, as the root cause may be related to the differences in kernel execution and API-level interactions in the GPU as opposed to the CPU. This is because of the GPU computational architectures that behave fundamentally differently from the CPU. Simply said, the GPU may require a custom API that can effectively interface between the optimisation strategy framework and the GPU kernel..

REFERENCES

Almeida, J. S., Huang, C., Nogueira, F. G., Bhatia, S., & de Albuquerque, V. H. C. (2022). EdgeFireSmoke: A Novel Lightweight CNN Model for Real-Time Video Fire–Smoke Detection. *IEEE Transactions on Industrial Informatics*, *18*(11), 7889–7898. https://doi.org/10.1109/TII.2021.3138752

Baller, S. P., Jindal, A., Chadha, M., & Gerndt, M. (2021). *DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices*. http://arxiv.org/abs/2108.09457

Bhagat, S., Kokare, M., Haswani, V., Hambarde, P., Taori, T., Ghante, P. H., & Patil, D. K. (2024). Advancing real-time plant disease detection: A lightweight deep learning approach and novel dataset for pigeon pea crop. *Smart Agricultural Technology*, *7*. https://doi.org/10.1016/j.atech.2024.100408

Bonam, J., Kondapalli, S. S., Prasad, N. L. V., & Marlapalli, K. (2023). Lightweight CNN Models for Product Defect Detection with Edge Computing in Manufacturing Industries. *Journal of Scientific and Industrial Research*, *82*(4), 418–425. https://doi.org/10.56042/jsir.v82i04.72390

Bragagnolo, A., & Barbano, C. A. (2022). Simplify: A Python library for optimizing pruned neural networks. *SoftwareX*, *17*. https://doi.org/10.1016/j.softx.2021.100907

Burra, L. R., Karuna, A., Tumma, S., Marlapalli, K., & Tumuluru, P. (2023). MobileNetV2-based Transfer Learning Model with Edge Computing for Automatic Fabric Defect Detection. *Journal of Scientific and Industrial Research*, *82*(1), 128–134. https://doi.org/10.56042/jsir.v82i1.69928

Chang, S., & Zheng, B. (2024). A lightweight convolutional neural network for automated crack inspection. *Construction and Building Materials*, *416*. https://doi.org/10.1016/j.conbuildmat.2024.135151

Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., & Peter Campbell, J. (2020). Introduction to machine learning, neural networks, and deep learning. *Translational Vision Science and Technology*, *9*(2). https://doi.org/10.1167/tvst.9.2.14

Dorafshan, S., Thomas, R. J., & Maguire, M. (n.d.). *SDNET2018: An annotated image dataset for non-contact concrete crack detection using deep convolutional neural networks*. https://doi.org/10.15142/T3TD19

Gang, H., Guanglei, S., Xiaofeng, W., & Jinlin, J. (2023). CCNNet: a novel lightweight convolutional neural network and its application in traditional Chinese medicine recognition. *Journal of Big Data*, *10*(1). https://doi.org/10.1186/s40537-023-00795-4

He, W., Yang, D., Peng, H., Liang, S., & Lin, Y. (2021). An efficient ensemble binarized deep neural network on chip with perception-control integrated. *Sensors*, *21*(10). https://doi.org/10.3390/s21103407

Hou, X., Guan, Y., Han, T., & Zhang, N. (2022). *DistrEdge: Speeding up Convolutional Neural Network Inference on Distributed Edge Devices*. http://arxiv.org/abs/2202.01699

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). *Searching for MobileNetV3*. http://arxiv.org/abs/1905.02244

Hussain, H., Tamizharasan, P. S., & Rahul, C. S. (2022). Design possibilities and challenges of DNN models: a review on the perspective of end devices. *Artificial Intelligence Review*, *55*(7), 5109–5167. https://doi.org/10.1007/s10462-022-10138-z

Ju, R.-Y., Lin, T.-Y., Jian, J.-H., & Chiang, J.-S. (2022). *Efficient Convolutional Neural Networks on Raspberry Pi for Image Classification*. http://arxiv.org/abs/2204.00943

Kalgaonkar, P., & El-Sharkawy, M. (2022). CondenseNeXtV2: Light-Weight Modern Image Classifier Utilizing Self-Querying Augmentation Policies. *Journal of Low Power Electronics and Applications*, *12*(1). https://doi.org/10.3390/jlpea12010008

Kamath, V., & Renuka, A. (2023). Deep learning based object detection for resource constrained devices: Systematic review, future trends and challenges ahead. In *Neurocomputing* (Vol. 531, pp. 34–60). Elsevier B.V. https://doi.org/10.1016/j.neucom.2023.02.006

Kim, C. Y., Um, K. S., & Heo, S. W. (2023). A novel MobileNet with selective depth multiplier to compromise complexity and accuracy. *ETRI Journal*, *45*(4), 666–677. https://doi.org/10.4218/etrij.2022-0103

Kong, H., Liu, D., Huai, S., Luo, X., Subramaniam, R., Makaya, C., Lin, Q., & Liu, W. (2023). EdgeCompress: Coupling Multidimensional Model Compression and Dynamic Inference for EdgeAI. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *42*(12), 4657–4670.

https://doi.org/10.1109/TCAD.2023.3276938

Kristiani, E., Tsan, Y. T., Liu, P. Y., Yen, N. Y., & Yang, C. T. (2022). Binary and Multi-Class Assessment of Face Mask Classification on Edge AI Using CNN and Transfer Learning. *Human-Centric Computing and Information Sciences*, *12*. https://doi.org/10.22967/HCIS.2022.12.053

Li, S., Sun, Y., Yen, G. G., & Zhang, M. (2023). Automatic Design of Convolutional Neural Network Architectures under Resource Constraints. *IEEE Transactions on Neural Networks and Learning Systems*, *34*(8), 3832–3846. https://doi.org/10.1109/TNNLS.2021.3123105

Li, X., Xiang, Y., & Li, S. (2023). Combining convolutional and vision transformer structures for sheep face recognition. *Computers and Electronics in Agriculture*, *205*. https://doi.org/10.1016/j.compag.2023.107651

Liu, J., Tripathi, S., Kurup, U., & Shah, M. (2020). *Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications: A Survey*. http://arxiv.org/abs/2005.04275

Liu, Y., Fan, K., Wu, D., & Zhou, W. (2023). Filter pruning by quantifying feature similarity and entropy of feature maps. *Neurocomputing*, *544*. https://doi.org/10.1016/j.neucom.2023.126297

Lu, F., Zhang, Z., Guo, L., Chen, J., Zhu, Y., Yan, K., & Zhou, X. (2022). HFENet: A lightweight hand-crafted feature enhanced CNN for ceramic tile surface defect detection. *International Journal of Intelligent Systems*, *37*(12), 10670–10693. https://doi.org/10.1002/int.22935

Luo, Y., Cai, X., Qi, J., Guo, D., & Che, W. (2023). FPGA–accelerated CNN for real-time plant disease identification. *Computers and Electronics in Agriculture*, *207*. https://doi.org/10.1016/j.compag.2023.107715

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). *ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design*. http://arxiv.org/abs/1807.11164

Manvikar, A., & Reddy, P. (2023). *Dataset of groundnut plant leaf images for classification and detection*. https://doi.org/http://doi.org/10.17632/22p2vcbxfk.3

Mocerino, L., & Calimera, A. (2021). Fast and Accurate Inference on Microcontrollers with Boosted Cooperative Convolutional Neural Networks (BC-Net). *IEEE Transactions on Circuits and Systems I: Regular Papers*, *68*(1), 77–88. https://doi.org/10.1109/TCSI.2020.3039116

Narasimha Prasad, L. V., Dokku, D. B., Talasila, S. L., & Tumuluru, P. (2023). Hyper

Parameter Optimization for Transfer Learning of ShuffleNetV2 with Edge Computing for Casting Defect Detection. *Journal of Scientific and Industrial Research*, *82*(2), 171–177. https://doi.org/10.56042/jsir.v82i2.70250

Naveen, S., Kounte, M. R., & Ahmed, M. R. (2021). Low Latency Deep Learning Inference Model for Distributed Intelligent IoT Edge Clusters. *IEEE Access*, *9*, 160607–160621. https://doi.org/10.1109/ACCESS.2021.3131396

Neogi, D., Chowdhury, M. A., Akter, M. M., & Hossain, M. I. A. (2024). Mobile detection of cataracts with an optimised lightweight deep Edge Intelligent technique. *IET Cyber-Physical Systems: Theory and Applications*. https://doi.org/10.1049/cps2.12083

Özgenel, Ç. F. (2019). *Concrete Crack Images for Classification*. https://doi.org/10.17632/5y9wdsg2zt.2

Qin, L. W., Ahmad, M., Ali, I., Mumtaz, R., Zaidi, S. M. H., Alshamrani, S. S., Raza, M. A., & Tahir, M. (2021). Precision Measurement for Industry 4.0 Standards towards Solid Waste Classification through Enhanced Imaging Sensors and Deep Learning Model. *Wireless Communications and Mobile Computing*, *2021*, 1–10. https://doi.org/10.1155/2021/9963999

Rani, S. D., Burra, L. R., Kalyani, G., & Rao, N. K. B. (2023). Edge Intelligence with Light Weight CNN Model for Surface Defect Detection in Manufacturing Industry. *Journal of Scientific and Industrial Research*, *82*(2), 178–184. https://doi.org/10.56042/jsir.v82i2.69945

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. http://arxiv.org/abs/1801.04381

Sharma, A. K., Rana, A., & Kim, K. K. (2021). Lightweight image classifier for CIFAR-10. *Journal of Sensor Science and Technology*, *30*(5), 286–289. https://doi.org/10.46670/JSST.2021.30.5.286

Sharma, N., Sharma, R., & Jindal, N. (2021). Machine Learning and Deep Learning Applications-A Vision. *Global Transitions Proceedings*, *2*(1), 24–28. https://doi.org/10.1016/j.gltp.2021.01.004

Shehada, D., Turky, A., Khan, W., Khan, B., & Hussain, A. (2023). A Lightweight Facial Emotion Recognition System Using Partial Transfer Learning for Visually Impaired People. *IEEE Access*, *11*, 36961–36969. https://doi.org/10.1109/ACCESS.2023.3264268

Shi, C., Hao, Y., Li, G., & Xu, S. (2023). EBNAS: Efficient binary network design for image classification via neural architecture search. *Engineering Applications of Artificial Intelligence*, *120*. https://doi.org/10.1016/j.engappai.2023.105845

Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. http://arxiv.org/abs/1409.1556

Song, M., & Guo, Q. (2022). Efficient 3D object recognition in mobile edge environment. *Journal of Cloud Computing*, *11*(1). https://doi.org/10.1186/s13677-022-00359-6

Stahl, R., Hoffman, A., Mueller-Gritschneder, D., Gerstlauer, A., & Schlichtmann, U. (2021). DeeperThings: Fully Distributed CNN Inference on Resource-Constrained Edge Devices. *International Journal of Parallel Programming*, *49*(4), 600–624. https://doi.org/10.1007/s10766-021-00712-3

Thakuria, A., & Erkinbaev, C. (2024). Real-time canola damage detection: An end-to-end framework with semi-automatic crusher and lightweight ShuffleNetV2_YOLOv5s. *Smart Agricultural Technology*, *7*, 100399. https://doi.org/10.1016/j.atech.2024.100399

Thwal, C. M., Nguyen, M. N. H., Tun, Y. L., Kim, S. T., Thai, M. T., & Hong, C. S. (2024). OnDev-LCT: On-Device Lightweight Convolutional Transformers towards federated learning. *Neural Networks*, *170*, 635–649. https://doi.org/10.1016/j.neunet.2023.11.044

Tran, N. C., Pham, B.-T., Chu, V. C.-M., Li, K.-C., Le, P. T., Chen, S.-L., Frisky, A. Z. K., Li, Y.-H., & Wang, J.-C. (2024). Zero-FVeinNet: Optimizing Finger Vein Recognition with Shallow CNNs and Zero-Shuffle Attention for Low-Computational Devices. *Electronics*, *13*(9), 1751. https://doi.org/10.3390/electronics13091751

Wang, B., Ma, F., Ge, L., Ma, H., Wang, H., & Mohamed, M. A. (2021). Icing-EdgeNet: A Pruning Lightweight Edge Intelligent Method of Discriminative Driving Channel for Ice Thickness of Transmission Lines. *IEEE Transactions on Instrumentation and Measurement*, *70*. https://doi.org/10.1109/TIM.2020.3018831

Wang, F., Zhang, M., Wang, X., Ma, X., & Liu, J. (2020). Deep Learning for Edge Computing Applications: A State-of-the-Art Survey. *IEEE Access*, *8*, 58322–58336. https://doi.org/10.1109/ACCESS.2020.2982411

Wang, S., Pu, Z., Li, Q., & Wang, Y. (2022). Estimating crowd density with edge

intelligence based on lightweight convolutional neural networks. *Expert Systems with Applications*, *206*. https://doi.org/10.1016/j.eswa.2022.117823

Xia, M., Huang, Z., Tian, L., Wang, H., Victor, C., Zhu, Y., & Feng, S. (2021). SparkNoC: An energy-efficiency FPGA-based accelerator using optimized lightweight CNN for edge computing. *Journal of Systems Architecture*, *115*. https://doi.org/10.1016/j.sysarc.2021.101991

Xiao, F., Wang, H., Xu, Y., & Shi, Z. (2024). A Lightweight Detection Method for Blueberry Fruit Maturity Based on an Improved YOLOv5 Algorithm. *Agriculture (Switzerland)*, *14*(1). https://doi.org/10.3390/agriculture14010036

Xie, Q., Wu, M., Bao, J., Zheng, P., Liu, W., Liu, X., & Yu, H. (2023). A deep learning-based detection method for pig body temperature using infrared thermography. *Computers and Electronics in Agriculture*, *213*. https://doi.org/10.1016/j.compag.2023.108200

Xu, J., Zhao, Y., & Xu, F. (2022). RDPNet: a single-path lightweight CNN with re-parameterization for CPU-type edge devices. *Journal of Cloud Computing*, *11*(1). https://doi.org/10.1186/s13677-022-00330-5

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. In *Insights into Imaging* (Vol. 9, Number 4, pp. 611–629). Springer Verlag. https://doi.org/10.1007/s13244-018-0639-9

Yoo, T., Lee, S., & Kim, T. (2021). Dual image-based cnn ensemble model for waste classification in reverse vending machine. *Applied Sciences (Switzerland)*, *11*(22). https://doi.org/10.3390/app112211051

Yu, J., Zhou, G., Zhou, S., & Yin, J. (2021). A lightweight fully convolutional neural network for sar automatic target recognition. *Remote Sensing*, *13*(15). https://doi.org/10.3390/rs13153029

Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., & Lee, B. (2022). A survey of modern deep learning based object detection models. In *Digital Signal Processing: A Review Journal* (Vol. 126). Elsevier Inc. https://doi.org/10.1016/j.dsp.2022.103514

APPENDICES

APPENDIX 1

Table of Training Loss for 35 Epochs using Transfer Learning

Epoch	MobileNet V2	MobileNet V3-Small	MobileNet V3-Large	ShuffleNet V2x0.5	ShuffleNet V2x1.0	ShuffleNet V2x1.5	ShuffleNet V2x2.0	VGG-16
1	2.15	2.15	2.81	0.96	1.10	1.43	1.51	1.39
2	1.68	1.62	1.77	0.70	0.83	1.10	1.19	0.85
3	1.59	1.55	1.71	0.62	0.68	0.86	0.91	0.58
4	1.56	1.52	1.64	0.58	0.59	0.71	0.76	0.53
5	1.47	1.50	1.60	0.53	0.54	0.63	0.63	0.45
6	1.43	1.49	1.61	0.53	0.50	0.56	0.57	0.43
7	1.38	1.52	1.64	0.50	0.49	0.53	0.54	0.40
8	1.34	1.43	1.59	0.49	0.45	0.49	0.50	0.38
9	1.32	1.41	1.48	0.48	0.44	0.46	0.48	0.34
10	1.30	1.42	1.47	0.47	0.43	0.45	0.45	0.34
11	1.27	1.51	1.47	0.46	0.42	0.45	0.43	0.31
12	1.23	1.48	1.54	0.46	0.43	0.43	0.44	0.32
13	1.22	1.44	1.53	0.47	0.41	0.43	0.41	0.30
14	1.18	1.33	1.48	0.49	0.43	0.42	0.41	0.29
15	1.15	1.32	1.37	0.45	0.40	0.40	0.40	0.29
16	1.12	1.27	1.38	0.46	0.40	0.41	0.40	0.28
17	1.07	1.27	1.37	0.45	0.40	0.40	0.39	0.26
18	1.04	1.26	1.37	0.45	0.41	0.39	0.40	0.25
19	1.02	1.26	1.37	0.45	0.40	0.40	0.39	0.24
20	1.00	1.25	1.34	0.45	0.40	0.40	0.39	0.25
21	0.98	1.24	1.34	0.44	0.40	0.39	0.39	0.25
22	0.98	1.21	1.25	0.45	0.39	0.40	0.38	0.23
23	0.97	1.16	1.26	0.44	0.40	0.38	0.39	0.23
24	0.97	1.15	1.23	0.43	0.39	0.40	0.39	0.22
25	0.92	1.12	1.20	0.45	0.40	0.38	0.40	0.21
26	0.90	1.09	1.17	0.44	0.40	0.39	0.38	0.20
27	0.89	1.06	1.14	0.43	0.40	0.38	0.39	0.22
28	0.86	1.07	1.12	0.44	0.39	0.40	0.38	0.22
29	0.84	1.08	1.14	0.45	0.39	0.40	0.38	0.21
30	0.84	1.08	1.11	0.43	0.38	0.40	0.38	0.19
31	0.65	0.82	0.88	0.29	0.26	0.25	0.26	0.15
32	0.59	0.73	0.80	0.26	0.23	0.22	0.22	0.12
33	0.57	0.72	0.78	0.23	0.20	0.20	0.20	0.11
34	0.55	0.68	0.74	0.23	0.19	0.19	0.19	0.11
35	**0.54**	**0.67**	**0.73**	**0.23**	**0.19**	**0.19**	**0.18**	**0.11**

APPENDIX 2

Table for Validation Loss for 35 Epochs using Transfer Learning

Epoch	MobileNet V2	MobileNet V3-Small	MobileNet V3-Large	ShuffleNet V2x0.5	ShuffleNet V2x1.0	ShuffleNet V2x1.5	ShuffleNet V2x2.0	VGG-16
1	1.68	1.80	1.95	0.70	0.94	1.13	1.20	1.13
2	1.73	1.44	2.25	0.65	0.61	0.76	0.83	0.75
3	1.45	1.39	1.93	0.79	0.53	0.83	0.69	0.60
4	1.40	1.46	1.84	0.72	0.62	0.84	0.50	0.45
5	1.33	1.56	1.42	0.91	0.55	0.95	1.11	0.41
6	1.25	1.57	1.64	0.61	0.61	0.73	0.64	0.41
7	1.15	3.00	1.94	0.63	0.69	0.72	0.73	0.48
8	1.22	1.92	1.58	0.81	0.57	0.76	0.79	0.41
9	1.21	1.54	1.95	0.72	0.58	0.64	0.76	0.50
10	1.14	**3.96**	1.65	0.86	0.43	0.79	0.59	0.60
11	1.14	2.38	1.61	0.66	0.79	0.69	0.57	0.50
12	1.08	1.96	2.22	0.69	0.53	0.83	0.65	0.49
13	1.04	1.61	1.79	0.70	0.79	0.67	1.01	0.43
14	1.13	1.52	1.36	0.93	0.56	0.56	0.71	0.55
15	1.00	1.26	1.40	0.83	0.56	0.63	0.74	0.47
16	1.04	1.41	1.63	0.89	0.58	0.67	0.65	0.46
17	0.92	1.49	1.31	0.71	0.77	0.62	1.05	0.31
18	0.85	1.26	1.51	0.61	0.75	0.74	0.75	0.45
19	0.85	1.27	1.39	0.83	0.91	0.77	0.73	0.47
20	0.75	1.18	1.60	0.72	0.66	0.56	0.74	0.50
21	0.79	1.48	1.81	0.59	0.52	0.54	0.58	0.33
22	0.73	1.21	1.13	0.88	0.66	0.84	0.78	0.45
23	0.86	1.25	1.16	0.79	0.84	0.81	0.53	0.39
24	1.02	1.16	1.28	0.87	0.59	0.88	0.55	0.43
25	0.71	1.93	1.46	0.79	0.60	1.00	0.56	0.34
26	0.86	1.22	1.96	0.70	0.86	0.71	0.82	0.56
27	0.94	1.36	1.31	0.63	0.63	0.83	0.60	0.42
28	0.89	1.37	1.07	0.74	0.68	0.55	0.72	0.53
29	0.87	0.91	1.96	0.67	0.64	0.72	0.59	0.32
30	0.89	1.43	1.02	0.69	0.64	1.14	0.90	0.40
31	0.49	0.60	0.61	0.54	0.50	0.53	0.64	0.32
32	0.47	0.56	0.72	0.58	0.59	0.53	0.58	0.29
33	0.51	0.57	0.63	0.60	0.57	0.58	0.58	0.29
34	0.55	0.55	0.57	0.58	0.64	0.53	0.67	0.28
35	**0.45**	**0.63**	**0.59**	**0.51**	**0.61**	**0.49**	**0.56**	**0.28**

APPENDIX 3

Table for Training Accuracy (%) for 35 Epochs using Transfer Learning

Epoch	MobileNet V2	MobileNet V3-Small	MobileNet V3-Large	ShuffleNet V2x0.5	ShuffleNet V2x1.0	ShuffleNet V2x1.5	ShuffleNet V2x2.0	VGG-16
1	24.33	24.94	19.30	64.60	67.12	42.46	38.90	42.36
2	29.39	28.35	20.85	75.12	76.83	57.48	53.10	67.25
3	32.83	32.17	23.92	77.83	80.16	68.48	66.37	79.87
4	34.05	33.13	27.40	79.15	81.96	74.34	72.71	81.71
5	37.12	32.69	29.11	80.93	82.65	77.62	77.64	84.52
6	38.74	34.12	28.78	80.76	83.48	79.88	79.58	85.38
7	41.73	33.45	27.49	81.90	83.44	81.14	80.87	86.02
8	43.56	37.14	30.05	82.25	84.27	82.63	82.55	86.75
9	44.70	38.43	36.21	83.08	84.49	83.51	82.84	88.22
10	46.94	39.27	36.60	82.96	84.65	83.70	83.58	88.43
11	48.06	34.48	36.82	**83.25**	**85.29**	**84.18**	**84.55**	89.31
12	49.93	36.32	32.26	83.02	84.96	84.85	84.17	89.22
13	51.78	37.46	32.91	83.12	85.22	84.82	85.45	89.66
14	52.80	43.31	35.73	82.70	84.65	84.66	85.47	90.44
15	54.26	43.63	42.09	84.17	85.89	85.78	85.87	89.91
16	56.67	46.27	42.34	83.31	85.75	85.22	85.71	90.65
17	58.73	46.35	42.71	83.93	85.71	85.93	86.44	90.86
18	60.14	47.99	42.68	83.52	85.49	85.97	85.79	91.49
19	61.33	47.47	42.66	83.99	85.80	86.02	86.11	91.57
20	62.27	47.81	44.95	83.92	86.14	85.83	86.62	91.40
21	62.89	48.38	44.54	84.13	85.38	86.23	85.85	91.35
22	63.06	50.76	50.01	83.81	85.51	85.85	86.24	92.11
23	63.75	53.24	49.14	83.93	85.47	86.28	86.44	92.26
24	63.34	54.19	50.97	84.47	85.72	85.91	86.32	92.57
25	65.64	55.80	52.63	84.14	85.99	86.60	85.44	92.64
26	66.41	56.61	53.85	84.10	85.54	86.03	86.55	93.10
27	66.72	59.15	55.02	84.40	85.98	86.41	86.24	92.69
28	68.33	58.59	56.25	84.10	86.02	85.66	86.44	92.72
29	69.12	57.81	55.34	83.54	85.99	85.77	86.54	92.73
30	68.87	57.94	57.37	84.28	86.07	85.74	86.37	93.26
31	76.13	69.00	67.14	89.50	90.87	91.35	90.83	95.13
32	78.38	72.78	70.17	90.91	92.09	92.46	92.44	95.76
33	79.40	73.56	71.08	91.75	92.73	92.99	92.95	95.94
34	79.92	74.82	72.22	91.81	93.16	93.55	93.40	96.01
35	**80.26**	**74.94**	**72.84**	**92.08**	**93.20**	**93.48**	**93.67**	**96.25**

APPENDIX 4

Table for Validation Accuracy (%) for 35 Epochs using Transfer Learning

Epoch	MobileNet V2	MobileNet V3-Small	MobileNet V3-Large	ShuffleNet V2x0.5	ShuffleNet V2x1.0	ShuffleNet V2x1.5	ShuffleNet V2x2.0	VGG-16
1	23.42	25.63	17.91	75.48	70.54	52.82	45.77	55.11
2	25.34	31.57	19.84	77.38	77.55	70.27	67.72	77.36
3	36.38	34.45	22.03	73.57	81.69	74.05	76.81	82.70
4	39.56	32.45	23.44	77.13	82.37	72.35	82.64	84.36
5	40.59	28.26	33.56	73.93	79.29	73.78	75.86	86.93
6	42.66	30.30	28.43	77.97	82.07	76.35	79.19	86.95
7	47.01	26.33	31.12	78.45	80.30	79.06	77.80	87.25
8	43.98	30.39	33.65	77.53	80.70	78.30	75.59	87.12
9	43.29	35.67	25.38	77.90	81.23	80.03	75.04	85.88
10	49.64	28.11	30.96	76.54	84.70	75.25	80.47	84.39
11	52.80	20.05	31.21	77.34	78.62	79.42	82.26	84.57
12	52.61	30.81	22.54	78.70	84.22	76.41	79.23	86.26
13	54.73	37.75	24.10	76.45	77.76	80.70	76.68	86.09
14	55.77	41.29	45.48	74.18	81.31	82.53	79.17	82.45
15	60.06	45.52	39.65	78.87	83.59	79.40	78.18	85.37
16	57.39	40.95	34.91	75.25	81.40	79.78	80.18	86.87
17	63.07	35.80	44.38	77.99	76.79	82.58	77.02	89.98
18	66.96	43.41	36.55	79.86	77.76	76.35	78.70	86.66
19	66.75	41.35	42.80	76.20	75.36	79.40	79.04	87.86
20	71.09	49.66	33.38	79.04	80.20	80.32	81.25	85.67
21	69.68	45.10	33.94	78.41	81.48	82.15	82.74	87.96
22	71.68	50.69	51.58	72.85	82.98	75.53	80.11	87.44
23	68.98	51.16	50.27	79.38	76.68	79.25	83.16	86.20
24	63.99	50.67	45.77	75.27	79.67	75.40	81.82	84.60
25	74.28	44.02	42.70	77.10	80.58	79.80	82.77	88.53
26	70.56	51.39	39.81	78.37	75.78	82.24	78.41	84.70
27	65.36	50.19	45.92	78.14	81.27	79.31	82.89	86.11
28	70.31	49.10	56.59	76.96	79.99	80.47	77.59	85.27
29	68.24	61.53	39.39	75.59	78.94	77.00	81.86	88.53
30	69.74	48.78	62.69	77.84	81.08	74.10	75.74	87.98
31	83.10	77.38	74.58	82.85	85.12	85.10	83.92	89.08
32	83.96	79.08	71.13	82.85	85.08	86.13	85.54	89.46
33	82.20	80.74	75.19	82.72	85.44	86.11	85.33	89.65
34	79.84	81.00	77.08	83.14	84.89	86.07	84.87	89.35
35	**83.16**	**78.77**	**76.14**	**84.53**	**85.44**	**85.77**	**85.86**	**89.37**

APPENDIX 5

Table for Training Loss for 35 Epochs without using Transfer Learning

Epoch	MobileNet V2	MobileNet V3-Small	MobileNet V3-Large	ShuffleNet V2x0.5	ShuffleNet V2x1.0	ShuffleNet V2x1.5	ShuffleNet V2x2.0	VGG-16
1	2.01	1.66	1.79	1.71	1.62	1.64	1.91	1.75
2	1.45	1.46	1.43	1.34	1.31	1.28	1.38	1.54
3	1.39	1.34	1.35	1.26	1.20	1.11	1.29	1.43
4	1.33	1.22	1.28	1.12	1.03	0.94	1.19	1.38
5	1.28	1.17	1.22	1.01	0.87	0.81	1.05	1.26
6	1.30	1.09	1.15	0.92	0.82	0.71	0.91	1.14
7	1.22	1.01	1.10	0.87	0.72	0.64	0.81	1.14
8	1.16	0.97	1.07	0.81	0.66	0.59	0.73	0.98
9	1.11	0.95	1.02	0.78	0.63	0.56	0.66	0.90
10	1.07	0.93	0.98	0.74	0.59	0.54	0.62	0.84
11	1.04	0.89	0.98	0.71	0.57	0.54	0.60	0.79
12	0.98	0.85	0.94	0.69	0.53	0.52	0.56	0.73
13	0.96	0.82	0.95	0.66	0.53	0.50	0.54	0.70
14	0.94	0.83	0.92	0.64	0.52	0.50	0.53	0.68
15	0.93	0.84	0.91	0.62	0.51	0.50	0.50	0.64
16	0.89	0.81	0.87	0.61	0.50	0.48	0.51	0.62
17	0.88	0.78	0.86	0.59	0.49	0.48	0.50	0.61
18	0.86	0.75	0.86	0.59	0.48	0.47	0.47	0.57
19	0.83	0.73	0.81	0.56	0.49	0.46	0.47	0.54
20	0.84	0.73	0.80	0.58	0.47	0.47	0.48	0.53
21	0.81	0.73	0.79	0.58	0.48	0.46	0.46	0.51
22	0.80	0.72	0.79	0.54	0.49	0.46	0.46	0.50
23	0.79	0.70	0.76	0.55	0.47	0.46	0.47	0.50
24	0.79	0.69	0.74	0.55	0.49	0.45	0.46	0.46
25	0.76	0.69	0.75	0.52	0.48	0.46	0.45	0.46
26	0.75	0.69	0.71	0.55	0.46	0.46	0.45	0.44
27	0.75	0.66	0.73	0.53	0.46	0.44	0.45	0.44
28	0.75	0.65	0.71	0.53	0.46	0.44	0.44	0.43
29	0.73	0.67	0.70	0.52	0.48	0.45	0.46	0.41
30	0.70	0.65	0.69	0.51	0.46	0.45	0.45	0.40
31	0.51	0.42	0.46	0.37	0.31	0.31	0.29	0.31
32	0.45	0.37	0.40	0.33	0.28	0.27	0.26	0.29
33	0.44	0.35	0.39	0.32	0.26	0.26	0.25	0.28
34	0.42	0.33	0.37	0.31	0.26	0.25	0.24	0.27
35	**0.41**	**0.32**	**0.35**	**0.29**	**0.25**	**0.24**	**0.23**	**0.27**

APPENDIX 6

Table for Validation Loss for 35 Epochs without using Transfer Learning

Epoch	MobileNet V2	MobileNet V3-Small	MobileNet V3-Large	ShuffleNet V2x0.5	ShuffleNet V2x1.0	ShuffleNet V2x1.5	ShuffleNet V2x2.0	VGG-16
1	1.47	1.61	1.59	1.25	1.25	1.22	1.24	1.66
2	1.32	1.59	1.87	1.17	1.19	1.05	1.16	1.42
3	1.30	1.73	1.52	1.34	1.06	0.90	1.17	1.43
4	1.14	1.40	1.45	1.25	0.97	0.77	1.11	1.31
5	1.15	1.29	1.66	1.11	0.91	0.61	0.83	1.15
6	1.13	2.03	1.29	0.96	0.96	0.67	0.80	1.63
7	0.97	1.55	1.52	0.91	0.60	0.67	0.73	0.92
8	1.61	1.08	1.63	0.75	0.62	0.64	0.69	1.01
9	0.89	1.70	1.16	**3.21**	0.57	0.66	0.71	0.88
10	0.99	1.49	1.47	0.91	0.74	0.67	0.57	0.91
11	0.85	1.31	1.16	0.72	0.72	0.65	0.74	0.87
12	0.89	1.55	1.61	0.77	0.86	0.55	0.69	0.77
13	0.87	1.07	1.38	0.71	0.64	0.87	0.66	0.63
14	0.93	2.21	0.74	0.92	0.64	0.76	0.70	0.75
15	0.97	0.86	0.98	0.72	0.58	0.57	0.73	0.71
16	0.83	2.35	0.83	0.85	0.57	0.74	0.70	0.65
17	0.58	1.02	0.99	1.00	0.56	0.65	0.93	0.62
18	0.63	1.67	1.01	0.71	0.55	0.62	0.78	0.52
19	0.62	1.02	0.79	0.79	0.62	0.68	0.48	0.68
20	0.61	1.32	1.04	0.82	0.67	0.43	0.57	0.54
21	0.62	1.46	1.36	0.79	0.91	0.66	0.79	0.57
22	0.85	0.78	1.16	0.72	0.56	0.67	0.95	0.63
23	0.81	1.17	0.76	0.82	0.67	0.74	0.70	0.51
24	0.59	0.91	1.00	0.78	0.62	0.71	0.64	0.48
25	0.70	0.99	0.86	0.62	0.53	0.60	0.92	0.47
26	0.70	0.91	0.93	0.76	1.22	0.73	0.77	0.53
27	0.58	0.72	1.06	0.69	0.63	0.64	0.57	0.56
28	0.80	0.69	1.56	0.88	0.54	0.60	0.58	0.62
29	0.82	1.85	1.10	0.79	0.59	0.72	0.68	0.69
30	1.78	1.15	0.82	0.71	0.53	0.74	0.68	0.55
31	0.45	0.56	0.54	0.56	0.54	0.58	0.58	0.46
32	0.39	0.51	0.49	0.57	0.53	0.57	0.56	0.48
33	0.37	0.54	0.42	0.58	0.56	0.56	0.67	0.47
34	0.46	0.48	0.51	0.52	0.58	0.56	0.59	0.52
35	**0.41**	**0.48**	**0.52**	**0.58**	**0.55**	**0.49**	**0.61**	**0.47**

APPENDIX 7

Table for Training Accuracy (%) for 35 Epochs without using Transfer Learning

Epoch	MobileNet V2	MobileNet V3-Small	MobileNet V3-Large	ShuffleNet V2x0.5	ShuffleNet V2x1.0	ShuffleNet V2x1.5	ShuffleNet V2x2.0	VGG-16
1	27.99	31.16	29.09	33.99	35.21	36.19	32.33	21.99
2	38.30	37.73	39.44	43.42	44.05	47.19	41.23	29.46
3	41.72	44.95	43.71	49.17	51.72	55.98	46.38	33.45
4	43.68	51.63	47.72	55.59	60.91	64.86	52.14	36.96
5	47.02	54.85	51.53	61.26	67.86	70.33	59.74	43.79
6	46.39	58.97	55.06	65.04	69.70	74.10	66.11	52.08
7	50.20	62.65	57.46	68.11	73.47	77.04	70.50	53.78
8	54.51	64.57	59.33	70.00	75.96	78.66	73.31	61.14
9	56.62	66.08	61.39	71.66	77.22	80.10	76.02	65.43
10	58.13	66.86	64.00	73.11	78.79	80.47	77.29	68.25
11	60.45	68.48	63.62	74.30	79.57	80.76	78.75	70.77
12	63.03	69.83	65.03	75.55	81.24	81.64	79.98	73.14
13	63.77	70.58	65.69	76.22	81.22	82.14	80.68	74.44
14	65.19	70.80	66.87	76.87	81.56	82.52	81.20	75.28
15	65.38	70.44	67.78	78.19	82.12	82.26	82.34	76.71
16	67.21	71.61	69.02	78.17	82.09	82.75	81.69	78.25
17	67.69	72.71	69.15	78.57	82.77	82.70	82.27	78.49
18	68.27	74.41	69.40	78.92	82.99	83.63	83.30	80.05
19	69.39	74.42	71.20	79.75	82.77	83.44	82.71	80.88
20	69.50	74.66	71.68	79.24	83.23	83.71	83.13	81.50
21	70.37	74.75	72.05	78.99	82.72	83.91	83.53	81.90
22	71.12	75.07	72.06	80.96	82.60	83.73	83.35	82.23
23	71.28	75.67	73.40	79.97	83.25	83.93	83.07	82.81
24	71.30	76.44	73.95	80.25	82.92	83.96	83.65	83.96
25	72.32	76.16	73.68	81.42	82.82	83.67	83.84	84.16
26	72.77	76.24	74.90	80.11	83.60	84.03	83.66	84.50
27	73.24	76.78	73.93	81.11	83.61	84.66	83.70	84.97
28	72.71	77.61	74.86	80.85	83.85	84.70	84.52	85.00
29	74.03	77.05	75.59	81.84	83.43	84.22	84.12	85.75
30	74.64	77.91	75.60	81.47	83.70	84.23	83.84	86.24
31	81.86	84.98	83.63	86.84	88.68	89.15	89.59	89.22
32	83.86	86.56	85.38	88.17	90.06	90.27	90.69	90.10
33	83.97	87.22	85.76	88.71	90.71	90.77	91.23	90.65
34	84.95	87.80	86.03	88.81	90.68	91.13	91.52	90.92
35	**85.28**	**88.48**	**87.46**	**89.47**	**91.06**	**91.56**	**91.92**	**90.85**

APPENDIX 8

Table for Validation Accuracy (%) for 35 Epochs without using Transfer Learning

Epoch	MobileNet V2	MobileNet V3-Small	MobileNet V3-Large	ShuffleNet V2x0.5	ShuffleNet V2x1.0	ShuffleNet V2x1.5	ShuffleNet V2x2.0	VGG-16
1	40.19	26.77	31.99	45.88	45.08	44.13	41.79	22.45
2	44.59	34.39	32.47	48.46	47.60	56.38	49.45	33.04
3	36.47	39.29	37.73	56.52	56.14	62.02	53.05	36.93
4	47.77	47.85	42.30	53.81	69.59	69.72	57.72	38.57
5	51.66	46.53	46.93	61.34	74.49	75.97	68.16	51.85
6	52.08	43.83	52.46	67.63	70.35	76.58	71.57	39.31
7	62.71	55.93	45.48	68.24	80.20	76.91	73.84	64.02
8	44.19	55.68	50.82	75.25	78.56	79.67	75.93	61.51
9	63.89	47.64	57.68	68.98	80.24	78.96	75.48	65.30
10	61.20	53.66	51.05	73.82	76.64	77.17	79.50	66.18
11	68.52	55.37	58.65	77.48	76.83	78.18	75.40	68.46
12	65.38	54.36	54.99	76.39	75.17	80.43	77.65	71.99
13	65.55	61.49	57.20	76.37	80.35	76.98	77.48	77.06
14	64.65	40.55	72.14	71.15	77.90	75.84	80.43	76.39
15	60.40	71.65	66.52	78.03	80.05	80.53	75.74	75.78
16	67.21	56.99	69.99	74.66	79.97	77.31	77.44	77.59
17	78.91	65.07	69.07	69.78	84.60	78.66	74.60	78.32
18	75.17	57.13	65.82	77.15	81.65	79.59	78.81	81.82
19	77.08	69.97	72.37	74.03	80.43	77.76	82.70	75.46
20	76.49	63.47	67.05	73.57	79.15	84.81	80.30	82.34
21	78.20	57.28	55.41	77.00	74.79	80.32	77.48	79.02
22	69.82	74.85	66.77	76.47	81.46	81.02	75.76	79.50
23	70.20	66.08	71.19	74.07	82.09	78.20	80.62	82.13
24	77.67	68.96	65.03	75.00	79.06	78.32	79.00	84.60
25	74.56	66.14	68.18	80.18	82.24	80.81	73.74	84.24
26	74.83	69.74	68.86	75.67	67.76	79.97	77.61	81.97
27	79.31	74.64	61.13	75.76	78.32	80.13	80.43	84.15
28	70.01	75.06	61.97	72.64	82.83	81.06	79.46	80.18
29	72.26	53.91	58.56	74.22	81.42	79.44	79.78	80.24
30	65.05	66.90	74.56	76.24	81.10	78.43	77.78	84.36
31	84.87	82.77	83.27	82.34	84.93	84.07	84.55	86.81
32	87.31	83.23	84.20	82.60	85.06	84.81	84.89	86.15
33	87.08	83.67	85.86	82.83	84.34	84.62	84.22	86.64
34	85.92	83.90	84.76	82.77	85.61	85.86	86.15	86.26
35	**87.50**	**84.03**	**83.73**	**83.23**	**85.42**	**85.82**	**86.15**	**86.34**

APPENDIX 9

Sample Image Classification Dataset for Groundnut Plant Leaf Disease

		
a) Early Leaf Spot	c) Early Rust	e) Nutrition Deficiency
		
b) Late Leaf Spot	d) Rust	f) Healthy Leaf

APPENDIX 10

Sample Image Classification Inference on the Edge Devices with Labelling

		
a) Early Leaf Spot	c) Early Rust	e) Nutrition Deficiency
		
b) Late Leaf Spot	d) Rust	f) Healthy Leaf

AUTHOR'S PROFILE



Muhammad Abbas Abu Talib received the Bachelor of Engineering (Hons.) in Electrical and Electronic Engineering from the Universiti Teknologi MARA (UiTM), Permatang Pauh, Pulau Pinang in 2023. He can be contacted at email: mat.abbas.talib@gmail.com.

LIST OF PUBLICATIONS

Abu Talib, M. A., Setumin, S., Abu Bakar, S. J., Che Ani, A. I., & Cahyani, D. E. (2025). Systematic review of a lightweight convolutional neural network architectures on edge devices. *International Journal of Reconfigurable and Embedded Systems (IJRES),* 14(2), 339. https://doi.org/10.11591/ijres.v14.i2.pp339-352

Abu Talib, M. A., Setumin, S., Che Ani, A. I., & Abu Bakar, S. J. (2025). An Analysis of Lightweight Convolutional Neural Network Models for Image Classification Task on Edge Device. *2025 IEEE 15Th International Conference on Control System, Computing and Engineering (ICCSCE),* 175–180. https://doi.org/10.1109/iccsce65566.2025.11182642